MASTER EN SCIENCES INFORMATIQUES

La description synthétique de la stratégie d'apprentissage en EAO : une étude de cas en géométrie

Parent, André

Award date:
1990

Awarding institution:
Universite de Namur

Facultés
Universitaires
Notre-Dame de la Paix
Namur

Institut d'Informatique

La description synthétique de la stratégie d'apprentissage en EAO : une étude de cas en géométrie.

par André PARENT

Promoteur :
Mr. Claude CHERTON

Mémoire présenté
dans le cadre de l'obtention
du grade de licencié et maître
en informatique.

Année Académique 1989-1990

Résumé.

Dans ce mémoire, j'ai essayé de montrer que, grâce à la description synthétique de la stratégie d'apprentissage et à la compréhension de la matière par le programme, on pouvait construire de meilleurs didacticiels sans que cela ne demande un travail trop important. Ces didacticiels rempliront les conditions de flexibilité et d'adéquation.

Un ensemble d'outils fournis par un système auteur permettra de créer ces didacticiels. Ces outils devront être mis au point suite à une collaboration entre les professeurs et les concepteurs du système auteur et également suite à une longue expérimentation de didacticiels construits grâce à ces outils.

Abstract.

In this paper, I have tried to show that, with help of the synthetical description of the learning strategy and of the subject's understanding by the program, better educational programs could be built but without that implies not a too great work. Those educational programs will satisfy the conditions of flexibility and adequacy.

A set of tools supplied by an authoring system will allow to create those instructional systems. These tools will be built further to a collaboration between teachers and conceptors of the authoring system and also to a long experimentation of programs built with help of those tools.

Je remercie Monsieur C.Cherton pour les critiques, les renseignements et les conseils apportés tout au long de ce travail ainsi que pour sa disponibilité et sa patience.

TABLE DES MATIERES.

Introduction.

1. Le choix du sujet.

Au moment du choix du sujet de mon mémoire, j'ai été sensibilisé par les problèmes que peuvent rencontrer les enseignants dans leurs tentatives d'aboutir à une meilleure qualité de l'enseignement. C'est plus précisément un texte de B.S.BLOOM qui m'a poussé à choisir un tel sujet. Dans cet article, "Le défi des deux sigmas : trouver des méthodes d'enseignement collectif aussi efficace qu'un précepteur" [Bloom], B.S.BLOOM signale qu'il existe un grand écart entre les résultats obtenus par les apprenants d'une classe "normale" et les résultats d'apprenants recevant un enseignement individualisé. Pour réduire cet écart, B.S. BLOOM propose différentes solutions parmi lesquelles la solution "un apprenant/un enseignant" qui est bien entendu impensable.

En réfléchissant au problème, je me suis posé la question de savoir si l'informatique ne pouvait pas jouer un rôle positif dans la réduction de cet écart. C'est donc pour cette raison que je me suis dirigé vers l'enseignement assisté par ordinateur. Et après avoir collaboré avec Mr. Cherton, je suis certain de n'avoir pas fait fausse route en pensant que l'informatique a son rôle à jouer.

2. Un développement pas à pas.

Ce mémoire s'insère dans un projet de plus grande envergure dont le lecteur trouvera une description détaillée au chapitre 2. Pour réaliser ce projet dans son entièreté, plusieurs mémoires seraient nécessaires. Dès le début de ce travail, nous le savions et notre objectif a donc simplement été de montrer la faisabilité d'un tel projet.

3. Notre approche.

Le chapitre 1 fait un historique des techniques d'aide à l'enseignement. Le but de ce chapitre est de montrer que ces techniques ne sont pas nouvelles et qu'il existe un problème avec l'enseignement assisté par ordinateur.

Dans le chapitre 2, nous décrivons une classe de didacticiels pour laquelle la description des dialogues avec l'apprenant pourrait être donnée de façon synthétique, indépendante de chaque exercice particulier, tout en fournissant une analyse de qualité des réponses. Si cela s'avère effectivement possible, on pourrait espérer pouvoir définir des didacticiels de qualité avec un rendement nettement meilleur que celui des systèmes auteurs classiques. Il en découle la définition d'un projet-cadre : l'élaboration d'un système auteur relatif aux didacticiels de ladite classe.

Le chapitre 3 décrit le prototype réalisé dans le cadre de ce mémoire. Ce prototype est basé sur certains principes du projet-cadre et veut donc montrer que ces principes sont réalistes. Le chapitre 4 donne le mode d'utilisation de ce prototype.

Des améliorations et des modifications à apporter au prototype pour arriver à un didacticiel pratiquement utilisable constituent le chapitre 5.

Enfin, les annexes présentent les textes des procédures du prototype réalisé.

Chapitre 1 : L'enseignement assisté par ordinateur.

Par Enseignement Assisté par Ordinateur (E.A.O), j'entends toute utilisation de l'ordinateur comme outil pédagogique.

1.1. Les types d'EAO.

On peut classer les différents types d'E.A.O selon l'utilisation qu'on fait de l'ordinateur et selon Robert P. Taylor [TAYLOR], toute utilisation de l'ordinateur au niveau de l'enseignement tombe sous un des trois modes suivants : tutor, tool et tutee. Précisons un peu ce que cachent ces termes [PAGANO].

1. **Tutor** : l'ordinateur **dirige** l'apprenant et lui fournit la connaissance nouvelle. Il vérifie la compréhension d'une partie avant de passer à une partie plus complexe.

2. **Tool** : l'ordinateur **aide** l'apprenant dans son processus d'apprentissage mais ne dirige pas ses efforts. L'ordinateur est pour l'apprenant une aide qui lui permet de mener à bien son travail.

3. **Tutee** : l'ordinateur **est dirigé** par l'apprenant dans le sens où c'est l'apprenant qui décide de la manière à adopter pour assimiler une nouvelle matière.

Cette classification n'est certainement pas la seule existante et elle sera, comme vous pourrez le voir au chapitre 2, rejetée par le concept de description synthétique de la stratégie d'apprentissage. Voici une autre classification qui décrit les didacticiels d'aujourd'hui :

Actuellement, il existe différents types de didacticiels : la liste qui suit n'est pas exhaustive. Je veux simplement citer quelques utilisations typiques.

1. Les didacticiels qui proposent des exercices, les corrigent et évaluent les réponses de l'apprenant : ces didacticiels testent la connaissance d'une matière supposée connue par des exercices du type questionnaire à choix multiple. Il s'agit en fait de simples vérificateurs de réponses du type "bon ou mauvais".

2. Les didacticiels d'enseignement programmé qui contiennent de la matière à assimiler par l'apprenant, un contrôle de la connaissance par questions-réponses et une succession de modules suivant le niveau de réponses de l'élève.

3. Il existe des didacticiels de documentation, d'illustration : ces didacticiels se basent sur une matière déjà vue mais ne vérifient pas pour autant la connaissance de cette matière. Ils montrent des exemples, des applications. De ce fait, ils peuvent aider à arriver à une meilleure connaissance.

4. Les simulateurs de phénomènes physiques, chimiques... qui sont là pour remplacer l'expérience.

1.2. Deux qualités nécessaires aux systèmes d'enseignement.

Selon R.L.Pagano [PAGANO], les notions d'adéquation et de flexiblité sont des exigences essentielles pour des systèmes d'enseignement informatisé. Expliquons un peu ce que représentent ces notions.

1.2.1. La flexibilité.

La flexibilité d'un outil informatique utilisé par un enseignant peut être évaluée en mesurant l'effort que cet enseignant fait pour preparer une leçon et pour la modifier. On tiendra compte, pour cette évaluation, de l'ensemble des modifications possibles d'une leçon.

1.2.2. L'adéquation.

Un outil est adéquat si les concepts propres à l'enseignement sont inclus dans le système. Souvent, dans l'approche traditionnelle de l'EAO, le système n'a pas une telle connaissance et le professeur doit définir les propriétés des objets chaque fois qu'il y fait référence. L'outil sera encore plus adéquat si le professeur a la possibilité d'intégrer ses propres stratégies pédagogiques. Cette possibilité rendra également le système plus flexible.

1.3. Historique.

1.3.1. Avant l'ordinateur.

On pourrait croire que les techniques d'aide à l'enseignement faisant appel aux machines sont assez récentes. Or, il y a plus de 60 ans que PRESSEY, préoccupé par l'automatisation de l'évaluation, proposait sa première machine à enseigner. C'était une machine qui offrait, lors de présentations successives, la possibilité de proposer uniquement les items ratés au passage précédent. Il la dota par la suite d'un dispositif permettant de subordonner le passage à la question suivante au choix de la réponse correcte parmi un ensemble limité de possibilités. Il s'agissait en quelque sorte d'un questionnaire à choix multiple mécanisé.

Par la suite, on utilisa des versions simplifiées de cette machine (PUNCHBOARD, DRUM TUTOR) et STEPHENS rapporte que l'utilisation du DRUM TUTOR permit à un groupe expérimental d'obtenir des résultats significativement supérieurs à ceux obtenus par un groupe de contrôle. Cette machine constituait donc bien un moyen de diminuer "l'écart des deux sigmas" cité dans l'introduction.

En 1954, SKINNER met au point le SLIDER TEACHING BOX. Cette machine ne laisse l'apprenant passer à la question suivante que lorsqu'il maîtrise parfaitement la question actuelle. Contrairement à la machine de PRESSEY

où l'apprenant doit choisir la réponse parmi un ensemble de solutions proposées, il doit ici construire sa réponse. C'est ici que surgit le problème inhérent à de telles machines : la réponse construite de l'apprenant doit être strictement identique à celle prévue par le concepteur de la machine alors que bien souvent, il existe différentes manières de formuler une réponse. On donna le qualificatif de "linéaire" à la programmation de SKINNER. Dans un programme skinnérien, on se contente d'adapter le rythme de présentation des informations aux capacités de chaque apprenant.

Quant à CROWDER, il estime que l'individualisation de l'enseignement doit aller plus loin. Dans l'optique de CROWDER, il faut tenir compte du comportement antérieur de l'apprenant évalué au travers de la dernière réponse fournie par celui-ci. CROWDER met au point une telle machine en 1959, l'AUTOTUTOR MARK I. Avec cette machine, et contrairement à celles de PRESSEY et SKINNER, l'évaluation de la réponse joue non seulement un rôle important dans le renforcement de la réponse correcte, mais permet surtout de déterminer l'information qui sera présentée ultérieurement. C'est la première fois qu'on a la possibilité de réaliser des ajustements individualisés en cours d'apprentissage. Je reparlerai de cette caractéristique au point **5.2.** Pour réaliser cette fonctionnalité, CROWDER utilise des "branchements". De plus, il ne permettait que les réponses choisies. La programmation de CROWDER fut appelée programmation ramifiée.

Les différentes machines de cette époque sont souvent restées expérimentales et n'ont pas considérablement fait progresser l'individualisation de l'enseignement.

1.3.2. Les didacticiels aujourd'hui.

Il faut se rendre compte que ce n'est que dans les années soixante que l'ordinateur est arrivé dans le monde de l'éducation. A ce moment et durant de longues années encore, la programmation ramifiée influença l'enseignement assisté par ordinateur.

On peut retrouver la classification des didacticiels d'aujourd'hui au point **1.1.**

Le problème inhérent aux différents types de didacticiels actuels est que le temps de développement est très grand pour une qualité pédagogique discutable. De plus, dans les systèmes d'EAO classiques, ce sont souvent des unités d'enseignement préconçues qui composent la leçon. La séquence de ces unités est déterminée par des branchements basés sur un ensemble prédéfini de réponses possibles de l'apprenant. Les qualités d'adéquation et de flexibilité définies plus haut sont très rarement atteintes.

1.3.3. L'arrivée de l'intelligence artificielle.

Les programmes devenant de plus en plus complexes, les chercheurs ont essayé d'appliquer des techniques d'intelligence artificielle. La compréhension du langage naturel, certaines méthodes d'inférences, des applications d'intelligence artificielle telles la simplification algébrique, la preuve de théorème ont été utiliées en vue de rendre les programmes d'EAO plus intelligents et plus efficaces. On peut citer par exemple : SCHOLAR, le tuteur en géographie de Carbonell et Collins, SOPHIE, le tuteur s'occupant de perturbations électroniques de Brown et Burton.

Pour terminer, je tiens à faire une remarque à propos de la raison qui me pousse à vouloir que l'informatique ait sa place à l'école : bureautique, robotique, télématique, téléinformatique, autant de mots qui apportent chez certains un sentiment de peur. Par conséquent, il me semble important de démystifier l'informatique. Commençons donc à l'école, en utilisant l'ordinateur comme outil de la même manière que livres, tableaux et autres matériels didactiques. Il faut pour cela que ces outils n'exigent pas un effort trop important de la part de l'utilisateur. Beaucoup d'enseignants redoutent l'informatique et appréhendent son introduction à l'école car ils croient que l'informatique à l'école signifie la disparition de l'enseignant.

Chapitre 2 : Le projet-cadre.

L'enseignement fait intervenir, entre autres, deux éléments essentiels :

- les apprenants
- l'enseignant.

Ainsi, pour qu'un didacticiel soit une aide à l'enseignement, je pense qu'il doit pouvoir s'adapter aux besoins de ces deux composantes. De plus, pour qu'il soit adopté par les enseignants, il faut que son utilisation amène des avantages appréciables et qu'il n'entraîne pas une charge de travail supplémentaire trop importante.

2.1. Le problème.

Actuellement, il existe deux problèmes relatifs à l'EAO. Ces deux problèmes sont étroitement liés.

On peut décrire le premier de ces problèmes comme suit : les enseignants estiment que la qualité des didacticiels est discutable et les auteurs de ces derniers trouvent que le travail que requiert la conception de ces didacticiels est énorme. Ce problème est en quelque sorte double : si on veut augmenter la qualité, la quantité de travail requise devient plus importante; et si on veut diminuer le travail, la qualité, déjà insatisfaisante pour la plupart des enseignants, diminue encore.

Le second problème consiste en le fait que le professeur a très rarement l'occasion d'intégrer sa stratégie pédagogique (sa manière de donner cours) dans un didacticiel qu'il veut utiliser. Quand il en a la possibilité, cela lui demande un gros travail supplémentaire. C'est souvent ce problème de manque de flexibilité qui décourage les enseignants d'utiliser l'ordinateur.

On peut résumer ce qui se passe actuellement avec l'EAO à ce que dit Carbonell :

> *"In most CAI systems, the computer does little more than what a programmed textbook can do, and one may wonder why the machine is used at all...When teaching sequences are extremely simple, perhaps trivial, one should consider doing away with the computer, and using other devices or techniques more related to the task." [Carbonell]*

2.2. Prévoir l'imprévisible.

A mes yeux, une des raisons de cet état de faits est qu'actuellement, la plupart des didacticiels et des outils à concevoir ces didacticiels reposent sur le concept de cours programmé. Tentons de définir ce concept : un cours programmé est un cours où l'on a essayé de prévoir explicitement une réaction adéquate à toute réponse de l'apprenant. Voici une phrase résumant bien ce que fait un cours programmé : *"The courseware author attempts to* ***anticipate every wrong response****, prespecifying branches to appropriate remedial material based on his ideas about what the underlying misconceptions might be that would cause each wrong response" [Barr & Feigenbaum].* Autrement dit, pour concevoir un cours programmé, on travaille non pas de manière générale mais bien en envisageant un maximum de cas particuliers. Dans un tel cours, on trouve une description très précise -description que je qualifierai d'analytique- de ce qu'il faut faire pour chaque réponse envisagée. On s'aperçoit que le travail de préparation d'un tel cours est très important et cela indépendamment du fait qu'il soit informatisé ou non. Cette description analytique nécessite une analyse très fine et donc très longue ainsi que l'utilisation systématique d'une rubrique "autres réponses" qui est peu discriminatoire et d'intérêt pédagogique très faible. L'utilisation d'une telle rubrique est indispensable car il est impossible de penser cas par cas à toutes les réponses possibles de l'apprenant.

Il me semble donc que pour remédier à cette situation, il faut supprimer, sinon en tout, au moins en partie la description analytique du scénario

d'apprentissage. L'objet de ce chapitre sera d'exposer l'approche que j'ai suivie.

2.3. But de la solution : construire un système auteur.

A terme, on peut espérer arriver à un système auteur à l'aide duquel il sera possible de construire des didacticiels de la classe décrite dans ce chapitre. Ce système devrait être caractérisé par une bonne qualité pédagogique et une efficacité considérablement meilleure que celle des systèmes actuels.

2.4. Contexte de travail des didacticiels répondant au projet-cadre.

Les didacticiels qui pourront être construits à l'aide du système-auteur permettront d'aider à résoudre des problèmes relatifs à une théorie supposée connue et consistant à mettre en oeuvre d'une démarche où plusieurs règles sont appliquées successivement pour passer de l'énoncé du problème à sa solution. Ainsi, l'activité didactique sera un entraînement à la mise en oeuvre de cette théorie. S'il est clair que certaines matières se prêtent mieux que d'autres à cette démarche telles que, par exemple, la physique, les mathématiques, ce sera surtout la stratégie pédagogique mise en place par le professeur qui influencera le choix de l'utilisation d'un tel didacticiel. Si le professeur insiste sur la mémorisation, il y a peu de chances que le didacticiel construit à l'aide du système auteur soit d'une grande utilité. Par contre, si l'accent est mis sur l'application de raisonnements, un tel didacticiel est applicable.

2.5. Caractéristiques de la solution du projet-cadre.

La notion de description synthétique est très importante dans la solution envisagée. Le qualificatif "synthétique" signifie que le professeur peut, en utilisant des outils préconçus, créer un didacticiel sans que cela ne lui demande un trop grand effort (toutes les interactions avec l'apprenant ne sont pas prévues dans le détail et prédéterminées) et sans pour cela que ce didacticiel ne réponde pas aux qualités d'adéquation et de

flexibilité. Pour bien comprendre ce que signifie ne pas être décrit en détail, voici un petit exemple :

> Dans une description analytique, si l'on veut enseigner la notion de carré d'un nombre, on donnera une table de carrés à l'apprenant. Par contre, dans une description synthétique, on donnera simplement la formule $y = x^2$. La description analytique demandera plus de temps et sera, même si elle parait plus détaillée, moins précise que la description synthétique car dans cette dernière, tous les cas sont prévus, ce qui n'est pas nécessairement vrai dans la description analytique.

Cette description synthétique est importante si l'on veut arriver à des didacticiels qui ne se contentent pas uniquement de dire si la réponse de l'apprenant est correcte après être allé vérifier dans une liste de réponses prévues. Si on veut aller plus loin dans l'optique d'un cours programmé, la description du scénario devient gigantesque. Dans l'optique de la description synthétique, on réagira par type d'événement pouvant survenir lors de la résolution d'un problème et ce qui sera prévu sera d'application pour tous les exercices.

Dans la solution, on veillera à ce que le professeur puisse modifier aisément cette description et de ce fait, puisse adapter le didacticiel non seulement à sa propre manière de donner cours mais aussi à chaque élève. De plus, donner la possibilité au professeur de pouvoir modifier certains comportements du didacticiel facilitera son adoption par l'enseignant.

2.6. Comment arriver à une telle solution ?

L'enseignement d'une matière quelconque peut être vu sous deux aspects :

- la matière proprement dite qui, dans beaucoup de cas, ne changera pas ou peu d'un professeur à un autre.
- la stratégie pédagogique d'aide à la résolution de problèmes dans la matière considérée.

Les didacticiels que l'on pourra construire avec le système auteur devront donc comporter ces deux parties bien distinctes. Cela nécessitera la présence de deux langages différents : l'un permettant la description de la connaissance envisagée et l'autre offrant au professeur la possibilité de décrire sa stratégie pédagogique.

On trouvera donc au minimum deux choses distinctes dans le didacticiel :

- d'une part, une description de la matière,
- d'autre part, une description de la stratégie d'apprentissage.

Ces deux descriptions nécessitent donc deux langages de description qu'il faudra, dans la mesure du possible rendre faciles d'utilisation. Voici un exemple possible de description de stratégie qui est écrit dans un langage pseudo-pascal :

```
DEBUT
  SI solution incorrecte
  ALORS
    DEBUT
      Discerner les parties correctes et incorrectes.
      Montrer à l'apprenant ces parties incorrectes.
      SI ce qui précède ne mène pas à une autocorrection
      ALORS
        DEBUT
          Demander la règle utilisée.
          Vérifier l'applicabilité.
          SI la règle est applicable
          ALORS
            DEBUT
              Trouver l'endroit où l'apprenant a
              probablement commis une erreur et montrer
              cette erreur.
            FIN
          SINON
            DEBUT
              Montrer la non applicabilité et proposer à
              l'apprenant une explication qui lui permettra
              de continuer.
            FIN
        FIN
    FIN
```

```
  SINON
    DEBUT
      SI solution correcte non terminale
      ALORS
        DEBUT
          Signaler à l'apprenant qu'il ne s'agit pas de la
          réponse finale.
          Lui demander de continuer en lui proposant une
          indication sur la nature de la réponse attendue.
        FIN
      SINON
        DEBUT
          SI solution correcte et terminale
            ALORS
              DEBUT
                Féliciter l'apprenant et l'encourager à
                poursuivre.
              FIN
        FIN
    FIN
FIN
```

Il s'agit bien entendu d'un exemple de stratégie. Cette description sera valable pour tous les exercices.

Cette description est très rudimentaire et ce sont les procédures que l'on utilise (discerner les parties correctes et incorrectes, ...) qui seront très importantes. De plus, il s'agit ici d'un prototype et il faut bien se rendre compte que l'on ne demandera pas au professeur, dans la version finale, de rédiger sa stratégie dans un tel langage.

Comme on peut s'en rendre compte dans la description qui précède, on doit pouvoir, face à une réponse d'un apprenant, analyser cette réponse sur base de la description de la connaissance et également pouvoir exploiter les résultats de cette analyse en tenant compte de la description de la stratégie d'apprentissage. Ces deux actions impliquent donc la présence de deux "programmes" :

- l'un, que j'appelerai analyseur de réponse, capable d'examiner une réponse. Cet analyseur pourrait générer, par exemple, les paramètres suivants : chemin probablement suivi par l'apprenant, réponse finale, réponse intermédiaire, utilisation

probable d'une règle, endroit où l'apprenant a probablement fait une erreur, règle qui semble mal comprise. Cette liste est non exhaustive et pour qu'elle réponde totalement aux désirs des professeurs, il faudra encore du temps, une bonne collaboration entre les concepteurs et les professeurs et une expérimentation du système auteur.

- l'autre, qui peut être appelé interpréteur ou compilateur, pouvant exploiter les résultats de l'analyseur de réponse sur base de la description de la stratégie pédagogique.

Dans le système auteur auquel on espère arriver à terme, ces deux "programmes" seront complètement indépendants de la description de la théorie et de la description de la stratégie et ils constitueront de ce fait la base du système auteur auquel on désire arriver.

On peut se poser la question de savoir ce qu'une telle découpe va amener aussi bien en avantages qu'en inconvénients.

Au niveau des avantages, il me semble pouvoir en comptabiliser trois :

1°. Description de la théorie.

Tout le monde sera d'accord avec le fait que la description de la matière envisagée sera moins souvent modifiée que la description de la stratégie d'apprentissage. De ce fait, l'investissement en temps nécessaire à la description de la théorie, même si il est grand, pourra être rentabilisé.

2°. Indépendance entre l'analyseur, l'interpréteur et les descriptions.

De par cette indépendance, on peut créer l'interpréteur et l'analyseur (qui constituent la base du système auteur) une fois pour toutes ce qui représente un gain de temps important. Le professeur

ne doit plus, comme c'est souvent le cas actuellement, s'occuper de programmation.

3°. Adéquation et flexibilité.

La description de la théorie qui est faite une seule fois offre un plus au niveau de l'adéquation du didacticiel. Le professeur n'aura plus à se soucier de la description de la théorie si ce n'est pour ajouter de nouvelles règles ou pour en modifier d'autres.

De même, la flexibilité d'un didacticiel construit avec le système auteur sera plus grande car le professeur aura la possibilité de changer totalement sa manière d'enseigner en modifiant une seule partie de la description de la stratégie d'apprentissage. Si l'on revient à l'exemple de la page 5, on se rend compte qu'en modifiant la partie du ALORS dans le cas de la solution incorrecte par la réaction suivante : "Dire à l'élève qu'il n'a rien compris", la manière de donner cours se trouvera totalement modifiée et cela pour tous les exercices.

Les partisans du cours programmé diront certainement que dans l'optique poursuivie dans ce travail, les réactions du programme face aux réponses de l'apprenant seront moins précises et orienteront donc moins bien l'apprenant dans son travail de résolution. Ce à quoi je réponds qu'en théorie, ils ont totalement raison mais en pratique, on se rend rapidement compte que prévoir toutes les réponses possibles de l'apprenant est pratiquement impossible. Dès lors dans un cours programmé, on devra, comme je l'ai déjà dit, utiliser une rubrique "Autres réponses". Par contre, en travaillant de manière synthétique, c'est à dire en prévoyant des réactions face à des types de réponses (ce qui est rendu possible en donnant au didacticiel la connaissance de la matière enseignée), une réponse appartiendra toujours à au moins un type et la réaction sera plus adéquate que celle prévue dans la rubrique "Autres réponses" de l'optique cours programmé.

2.7. Outils et avantages.

Le système auteur sera donc constitué d'un analyseur de réponses et d'un "interpréteur" de description de stratégie. Cet interpréteur se basera sur les paramètres renvoyés par l'analyseur pour appliquer la stratégie.

On doit, pour arriver à un outil intéressant, offrir au professeur (qui va décrire sa stratégie) un ensemble d'outils qu'il utilise lorsqu'il vérifie la connaissance d'une matière chaez un apprenant. Ce qui suit ne constitue pas une liste exhaustive mais plutôt un ensemble d'outils qui doit être présent pour que le système auteur soit adopté par les professeurs.

2.7.1. Les paramètres intéressants.

Comme je l'ai déjà dit, l'analyseur, après avoir examiné la réponse, renverra un certain nombre de paramètres sur lesquels l'interpréteur se basera pour appliquer la stratégie décrite par le professeur. Il se peut que l'interpréteur demande d'autres informations à l'analyseur. Toutes ces choses constituent donc des paramètres sur lesquels le professeur va pouvoir jouer dans sa description. Il en est quelques-uns qui sont évidents tels que correction de la réponse, temps de réponse. On peut également penser au fait de savoir si une règle particulière a été utilisée. Le professeur peut également vouloir que l'apprenant lui explique sa démarche. On le voit, pour arriver à un ensemble consistant, il faudra que les professeurs expliquent ce qu'ils veulent pouvoir décrire dans leur stratégie.

Un point sur lequel il me paraît important d'insister est la différence qui existe entre voir qu'une réponse est correcte et reconnaître une démarche d'un élève comme correcte. Un apprenant peut, par exemple, utiliser une bonne démarche sans que cela ne le mène à une réponse correcte soit parce qu'il a mal appliqué une règle, soit parce qu'il a fait une erreur de distraction. L'analyseur devra donc, face à une réponse, être capable de discerner les parties correctes de celles qui ne le sont pas. A partir de là, il pourra peut être

localiser dans la démarche la source des parties incorrectes.

2.7.2. Tenir compte du passé de l'apprenant.

"Often it is not sufficient to tell a student he is wrong and indicate the correct solution method. An intelligent CAI system should be able to make hypotheses based on a student's error history as to where the real source of his difficulty lies." [Koffman & Blount].

Ne pas pouvoir tenir compte du passé de l'apprenant n'est pas réaliste dans le cadre d'un didacticiel. Il faut comme le disent Koffman et Blount pouvoir se baser sur l'historique de l'apprenant. Voici un exemple qui montre cette importance : imaginons un apprenant qui n'a jamais pu donner une réponse correcte à des exercices d'un type particulier et qui, tout à coup, trouve la solution exacte d'un exercice de ce type. Le professeur, en tant qu'être critique, essaiera de savoir comment il s'est débrouillé pour arriver à cette réponse : soit l'apprenant a copié la réponse d'un autre élève, soit il a enfin compris la matière, soit il a eu de la chance,...

On peut penser à différentes solutions pour tenir compte du passé de l'apprenant :

1°. Utiliser une solution semblable à celle utilisée dans GUIDON. Dans cette solution, l'apprenant doit indiquer son niveau d'expertise en commençant sa session de travail. A partir de ce niveau , le didacticiel peut connaître le USE-HISTORY de l'apprenant : ce sont les règles que l'apprenant est censé connaître. A partir de cet ensemble de règles, le didacticiel pourra tenir compte des connaissances et des difficultés de l'apprenant et également évaluer ses progrès.

2°. Utiliser une solution avec identification. Dans cette solution, l'apprenant donne son nom avant de commencer à travailler. A chaque apprenant est attaché un ensemble d'informations sur son

passé. Cet ensemble sera constamment modifié par le didacticiel au cours de la session de travail. Ici, le programme pourra tenir compte des difficultés de l'apprenant et évaluer ses progrès de meilleure manière que dans la première solution. En effet, le didacticiel se basera sur un historique individuel plutôt que sur un historique probable établi sur base d'un niveau de connaissances comme dans GUIDON. Et l'on sait qu'un enseignement individualisé donne de meilleurs résultats qu'un enseignement généralisé.

La possibilité de tenir compte du passé de l'apprenant sera donc offerte au professeur. Ce sera lui, qui dans la description de la stratégie d'apprentissage, décidera de l'utiliser ou non.

2.7.3. Les métarègles.

L'être humain, face à un problème essaie souvent de se ramener à une situation plus simple. Pour arriver à cela, il réalise certaines transformations telles que par exemple, en géométrie, un changement de variables, une rotation. Il essaie de faire cette opération pour faciliter son processus de résolution et souvent, cela lui permet d'obtenir la réponse à son problème nettement plus rapidement et plus facilement. Il me semble bon de pouvoir apprendre aux élèves à utiliser de tels processus que l'on peut appeler métarègles. Le professeur doit donc avoir la possibilité d'intégrer l'utilisation de ces métarègles dans sa stratégie d'apprentissage.

La structure d'une métarègle serait : "Si vous êtes dans telle situation, alors faites la construction suivante et vous obtiendrez tel résultat". Ces métarègles constitueront donc une connaissance supplémentaire qui est souvent nécessaire pour aider un apprenant comme le dit Brown :

"These are just some of the problems which stem from the basic fact that teaching is a skill which requires knowledge additional to the knowledge comprising mastery of the subject domain." [Brown].

On peut envisager au moins deux manières d'offrir la possibilité d'utiliser ces métarègles, la première étant plus lourde que la seconde :

1°. Le professeur, après avoir introduit un exercice dans la base d'exercices, donne la métarègle et le moment d'application de celle-ci. Cette métarègle peut alors être considérée comme une donnée supplémentaire de l'exercice pour l'analyseur de réponses. Dans un premier temps, cette information supplémentaire ne sera pas communiquée à l'apprenant; c'est le professeur qui décidera, dans la stratégie d'apprentissage, de l'instant auquel elle sera communiquée à l'apprenant.

   Cette méthode est très spécifique car si deux exercices font appel à la même métarègle, le professeur devra rentrer cette règle deux fois ce qui peut représenter une perte de temps assez considérable et une lacune au niveau de l'adéquation. C'est pourquoi je crois que la seconde solution conviendra mieux.

2°. Dans un premier temps, un ensemble d'experts dans la matière envisagée collecte les différentes constructions avec leurs conditions d'application. Ensuite, on les introduit dans le didacticiel comme base de connaissances supplémentaires. L'analyseur, face à un exercice, pourra consulter ces métarègles afin de voir si il n'en existe pas une qui soit applicable au problème.

   Les métarègles, vues de cette manière, sont plus génériques. L'effort de départ (collecte des métarègles) sera plus important mais pourra être rentabilisé par le fait que cela enlèvera au professeur une part de travail.

Comme pour le fait de tenir compte du passé, ce sera le professeur qui décidera de l'utilisation de ces métarègles dans la description de sa stratégie.

2.7.4. Les explications spécifiques pour chaque règle.

Si on veut arriver à un système auteur permettant de construire des didacticiels qui soient une aide semblable à celle prodiguée par le professeur, on ne peut imaginer que les explications soient identiques quelles que soient les règles utilisées et les circonstances. En effet, un professeur ne donnera pas la même explication pour une règle simple que pour une règle compliquée. De même, s'il s'agit d'une règle qui est supposée maîtrisée, le professeur n'aura pas le même comportement que s'il s'agit d'une règle que l'apprenant vient d'apprendre.

Il faut donc que l'on puisse adapter les explications et les commentaires à chaque règle et aux circonstances d'application de chacune de ces règles. Néanmoins, ce sera le professeur qui décidera d'utiliser ou non cette fonctionnalité et il peut très bien décider que les explications seront toutes les mêmes.

Pour réaliser la première partie de cette fonctionnalité (explication spécifique à chaque règle), on peut imaginer que le professeur fournisse au didacticiel les explications à donner pour chaque règle ou pour chaque ensemble de règles. Au moment de donner l'explication, le programme ira consulter cette liste pour savoir quelle explication il doit présenter.

Pour pouvoir assurer une explication cohérente en fonction des circonstances, il me semble que l'on va devoir se baser sur le passé de l'apprenant. Le professeur aura la possibilité de donner différentes explications en fonction des circonstances et le didacticiel, se basant sur le passé de l'apprenant, choisira l'explication qui convient le mieux au moment.

Je le répète, il s'agit d'une possibilité qui sera offerte au professeur mais il se peut qu'il ne désire pas l'utiliser et donc donner la(les) même(s) explication(s) quelles que soient la règle et les circonstances.

2.7.5. Remédier au "Believe".

> *"It is advantageous for the system to be able to recognize alternative ways of solving problems,* ***including the incorrect methods that the student might use as a result of systematic misconceptions*** *about the problem or inefficient strategies." [Barr & Feigenbaum p.231].*

On peut donner du "believe" la définition suivante : il s'agit d'un ensemble de règles que l'apprenant croit être correctes mais qui, en réalité, ne le sont pas. Cela peut évidemment entraîner des erreurs dans la résolution de certains exercices. Comme le professeur peut essayer de corriger ce "believe", il est bon que le système auteur fournisse un outil capable de remédier ce problème. Le paragraphe suivant présente une idée de solution à laquelle j'ai pensé.

Cela se déroule en deux temps :

1°. Le professeur, après avoir introduit la matière à enseigner, rassemble toutes les règles faisant fréquemment l'objet de "believe" dans une base de données que l'on pourra appeler "BELIEVE".

2°. L'analyseur, en se basant sur le passé de chaque apprenant, pourra essayer de repérer les règles que chaque apprenant confond et ainsi créer un "believe" propre à chaque élève.

Cette base de donnée permettra, non seulement, de corriger l'apprenant mais aussi de réaliser ce qui est présenté au point 2.7.7.

Il s'agit bien entendu d'une idée de solution et il en existe très certainement d'autres.

2.7.6. Difficulté d'un chemin.

La difficulté, voilà bien un concept subjectif : ce qui est difficile pour l'un ne l'est pas forcément pour l'autre.

La mesure de la difficulté d'un chemin de résolution est complexe et il faut notamment tenir compte du degré de difficulté des règles utilisées et la longueur de ce chemin. Le degré de difficulté intervient dans cette mesure mais on ne peut envisager de mesurer la difficulté d'un chemin en sommant les différents degrés pour la raison suivante : si l'apprenant applique correctement une règle une fois, il devrait pouvoir le faire plusieurs fois; le degré de difficulté de cette règle reste le même mais sa difficulté d'application est modifiée.

De plus, il n'est pas rare qu'un apprenant qui a sa méthode de résolution pour un type d'exercice refuse de changer sa méthode pour une autre dont on lui dit qu'elle est plus simple. Souvent, pour l'apprenant, c'est la réponse qui compte et la manière pour y arriver importe peu alors qu'un chemin plus court témoigne d'une meilleure maîtrise de la matière. Il faut donc essayer de convaincre l'apprenant de changer de méthode car apprendre, c'est notamment modifier son comportement.

Pour résoudre ce problème de mesure, je propose l'idée de solution suivante : en utilisant la mémorisation du passé de l'apprenant, l'interpréteur pourra s'apercevoir du degré de connaissance de chaque règle théorique. On pourra donc calculer la difficulté du chemin parcouru.

Cette méthode a selon moi deux avantages :

1°. Le degré de difficulté de chaque règle ne sera pas une donnée complètement figée. Il changera avec la maîtrise de l'apprenant. Cela reflète plus la réalité professeur/apprenant.

2°. On calculera la difficulté en fonction d'une part, du chemin parcouru par l'apprenant et d'autre part, de son degré de maîtrise de la matière.

De nouveau, il s'agit d'une possibilité qui sera offerte au professeur. Si il désire que les apprenants appliquent sa méthode, il ne cherchera pas à savoir si la

méthode suivie par l'apprenant est plus ou moins difficile que la sienne.

2.7.7. Trouver le chemin probablement suivi par l'apprenant.

> *"A good teacher must understand what the student is doing, not just what he is supposed to do." [Barr & Feigenbaum p.231].*

Cette affirmation aura selon moi une répercussion importante sur la solution :

> Au niveau de la matière à enseigner, il faudra que la description corresponde non seulement à la théorie pure mais aussi et surtout à la manière dont l'apprenant voit cette matière. Les personnes qui décriront la connaissance devront donc faire appel à des experts dans cette matière et dans la manière de l'enseigner. Il me semble que le fait d'exprimer la matière selon la manière dont les apprenants la voient ne constitue pas un choix mais bien une nécessité.

Malgré cette précaution, il sera sans doute encore difficile de trouver le chemin suivi par l'apprenant dans certains cas. Ce n'est qu'après avoir utilisé un didacticiel conçu avec le système auteur que l'on pourra voir si la description de la matière a été bien réalisé.

> *"It is not assumed that the student reasons as the expert does, except for knowing less; the student's reasoning can, in fact, be substantially different from expert reasoning." [Barr & Feigenbaum p.231].*

Il faudra donc arriver à ce que l'expert, même si il ne raisonne pas comme l'apprenant, comprenne le raisonnement de celui-ci et puisse retrouver ce cheminement.

Outre les problèmes théoriques évoqués ci-dessus, il faut également penser aux interfaces qui permettront le dialogue avec l'apprenant. Il faudra que ces interfaces

soient simples d'utilisation et qu'ils répondent à l'attente des professeurs.

2.7.8. L'intégration du graphisme.

Dans certaines matières, il est bon d'avoir la possibilité de travailler de manière graphique. Il faut donc offrir cette possibilité au professeur.

Celui-ci pourra, selon son bon vouloir, intégrer ou non le graphisme. Cette intégration peut se faire de différentes manières et notamment :

1°. Lorsqu'un exercice est proposé à l'apprenant, on donne la représentation graphique du problème. L'apprenant peut ainsi visualiser le problème à résoudre. Il ne peut apporter aucune modification à cette représentation et doit répondre de manière textuelle.

2°. On se retrouve dans la même situation qu'au point 1° mais ici, lorsque l'apprenant répond, toujours de manière textuelle, la représentation se modifie en suivant ce que l'apprenant fait.

3°. On donne la possibilité à l'apprenant de répondre de manière graphique.

Encore une fois, il s'agit d'une possibilité qui est offerte au professeur et il peut très bien décider de s'en passer. C'est dans la description de la stratégie qu'il décidera de l'utiliser ou non.

On peut envisager que dans un premier temps, les représentations soient stockées dans une base que l'interpréteur irait consulter quand la description lui demanderait. Par après, on peut espérer arriver à ce qu'une partie du didacticiel puisse, en se basant sur une description graphique de la matière, construire les représentations graphiques.

2.8. Architecture d'un didacticiel.

Dans cette partie, je vais essayer de décrire l'architecture d'un didacticiel construit sur base du système auteur.

Deux parties se retrouveront sans aucune modification dans tous les didacticiels : il s'agit de l'analyseur de réponse et de l'interpréteur de description de stratégie. Pour qu'ils aient la possibilité d'effectuer correctement leur travail, ces deux composants nécessitent la présence d'autres parties qui sont :

1°. L'analyseur de réponse s'appuie sur la description de la matière à enseigner. Comme je l'ai dit plus haut, cette description ne sera pas une simple traduction de la théorie mais bien l'expression de la manière dont l'apprenant voit cette matière. De plus, il se peut que cette description soit double : l'une textuelle et l'autre graphique. Je l'appelerai **base de cours.**

2°. L'interpréteur de description de stratégie a, comme on peut s'en douter, besoin d'**une description de stratégie d'apprentissage.** Dans cette stratégie, le professeur pourra faire appel à différents outils (présentés au point 2.7) qui nécessiteront la présence d'autres composants. Je vais faire un relevé, non exhaustif, de ces composants :

   a) Pour tenir compte du passé de l'apprenant, on a besoin de son historique que j'appellerai **modèle de l'apprenant.** Ce modèle sera propre à chaque étudiant et pourra être constamment remis à jour par le didacticiel.

   b) Si le professeur désire utiliser des métarègles, il faudra qu'elles soient répertoriées dans une base de données que l'analyseur et l'interpréteur devront pouvoir aller consulter. J'appellerai cette base de données **base de métarègles.**

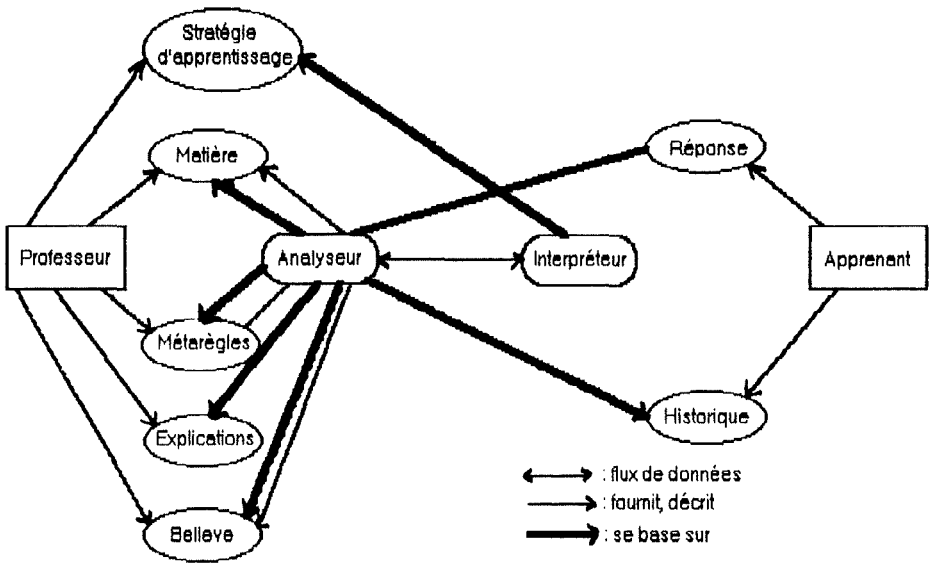
c) Le professeur qui décidera de donner des explications spécifiques aux circonstances et à l'apprenant devra, en collaboration avec le concepteur, rassembler ces explications dans une autre base de données accessible par l'interpréteur. Cette base de données sera appelée **base d'explications.**

d) En ce qui concerne le believe, si le professeur décide d'utiliser les erreurs communément commises par les apprenants, il faudra collecter ces erreurs et les stocker dans une autre base de données que l'on appelera **believe.**

e) Si le professeur décide d'utiliser l'outil graphique, cela nécessitera soit une description graphique de la matière combinée à un outil capable d'exploiter cette description, soit une base de données regroupant les différentes représentations graphiques.

f) On peut encore penser à un générateur d'exercices qui se baserait sur la(les) description(s) de la matière ou à une base de données regroupant les différents exercices à poser aux apprenants.

Sur un graphique, cela donnerait :



Chapitre 3 : Le prototype réalisé.

3.1. Le but du prototype.

Le but de ce prototype n'est pas d'arriver à un produit utilisable tel quel. J'ai voulu par ce prototype montrer deux choses :

1°. La faisabilité d'un système auteur qui s'inscrit dans l'optique présentée au chapitre 2.

2°. La possibilité d'écrire un didacticiel à partir du système auteur.

3.2. Quelle matière pour le prototype réalisé ?

3.2.1. Choix de la matière.

Comme je l'ai signalé précédemment au point **2.4**, certaines matières se prêtent mieux que d'autres à l'utilisation d'un didacticiel répondant à la description du projet-cadre. Dès lors, j'ai fait un choix parmi ces matières et j'ai opté pour la géométrie analytique car il s'agit d'une matière qui, non seulement répond aux critères décrits dans le chapitre 2, c'est à dire qui demande l'application de raisonnements pour la résolution d'exercices, mais aussi m'intéresse plutôt assez.

3.2.2. Une réduction de la matière.

Le but du mémoire n'était pas d'arriver à un produit utilisable tel quel dans l'enseignement mais bien de montrer la faisabilité de didacticiels répondant au projet-cadre et cela pour les raisons suivantes :

1°. Le temps disponible pour la réalisation du mémoire qui peut paraître très long mais qui en réalité est très court.

2°. De plus, il a fallu que je me familiarise avec un langage de programmation dont je ne connaissais que quelques rudiments : le langage PROLOG.

Ces raisons m'ont donc poussé à limiter la matière à ce qui est présenté au point 3.4.

3.3. La méthode de travail du programme résolveur.

Comme je l'ai signalé préalablement au point **2.6**, le didacticiel connaîtra certains objets de la matière à enseigner et des propriétés s'y rapportant grâce à la description de la matière.

Face à un exercice (composé de données et du but à atteindre), le programme résolveur aura le comportement suivant :

1°. Il cherchera les propriétés des données en entrée.
2°. Il tentera de trouver un théorème applicable à une ou plusieurs de ces propriétés.
3°. Il l'appliquera, ce qui aura pour effet de donner de nouvelles propriétés. Il recommencera le processus à l'étape 1° jusqu'au moment où il aura atteint le but.

Cette manière de faire résulte d'un choix que j'ai fait. J'aurais très bien pu prendre une autre option : l'analyseur attend que l'élève donne sa réponse et l'examine alors sur base de la matière pour obtenir des résultats exploitables par l'interprèteur de la description de la stratégie. Cette option se rapproche plus de la manière de procéder d'un professeur. Après réflexion, je me rends compte que mon choix aurait dû se porter sur cette manière de faire car, comme je viens de le dire, elle correspond plus à la manière de faire d'un professeur.

Dans l'état actuel du prototype, c'est ce programme résolveur qui renvoie une série de paramètres sur lesquels l'interpréteur se basera pour analyser la réponse de l'élève.

On constate que la démarche du programme résolveur n'est pas forcément la plus adéquate ni la plus rapide ni la plus facile pour les raisons suivantes :

1°. Ne pas être adéquate : si à un moment donné de l'exécution, on ne trouve plus aucun théorème applicable et que le but n'est pas atteint, que se passera-t-il ?
2°. Ne pas être la plus rapide ou la plus facile : comme je l'ai dit au point 2.7.3, souvent, l'être humain effectue des transformations qui lui permettent de trouver la réponse plus rapidement ou plus facilement. Ces transformations peuvent être par exemple dans le cas de la géométrie analytique un changement de coordonnées, une rotation, ... Le programme n'effectuant pas ces transformations, allongera donc sa réponse. On pourrait lui permettre ces transformations en ajoutant un ensemble de métarègles dont j'ai parlé au point 2.7.3.

3.4. Le choix du PROLOG et d'une station de travail.

3.4.1. Le choix du PROLOG.

Le lecteur ayant une petite expérience du PROLOG aura compris lors de la description de la méthode de travail du programme résolveur (**3.3**) que ce langage peut constituer la brique de base du raisonnement que j'ai adopté. Le PROLOG s'appuyera sur la base de règles décrivant les aspects théoriques pris en considération pour résoudre l'exercice qu'on lui présentera sous forme de but (au sens PROLOG du terme).

C'est pourquoi, en accord avec Mr. Cherton, j'ai décidé d'utiliser le langage PROLOG.

Pour que le didacticiel soit une aide réelle pour l'apprenant, les règles décrivant la matière devront, non pas être une fidèle traduction de la théorie présentée dans un livre, mais être conforme à la façon dont l'élève perçoit cette matière. J'ai déjà parlé de ce problème au point 2.7.7 et il me semble que cela constitue un problème assez important qui ne pourra être résolu qu'après avoir utilisé un certain nombre de fois le système auteur.

3.4.2. Le choix d'une station de travail.

Après avoir décidé d'utiliser PROLOG, j'ai réalisé des tests sur différentes machines telles que PC XT, PC AT et station de travail.

La raison pour laquelle j'ai abandonné le travail sur PC XT est la lenteur du PROLOG pour prouver un but.

Sur PC AT, cela se déroulait déjà nettement mieux mais c'était encore assez lent et de plus, le PC AT de l'Institut étant rarement disponible, il n'était pas concevable que je travaille sur cette machine.

Il ne me restait plus qu'à travailler sur une station de travail dont les performances sont très bonnes. Un problème de cette décision est qu'aujourd'hui, il n'y a presque aucune école disposant de ce type de matériel. Mais il faut bien se rendre compte qu'il s'agit ici d'un prototype qui veut montrer la faisabilité d'un didacticiel répondant au projet-cadre et non pas être utilisé en tant que tel dans l'enseignement. Une fois ce prototype réalisé, on pourrait imaginer de réaliser sur de plus petites machines des didacticiels basés sur les mêmes principes et qui soient suffisamment performants grâce à l'utilisation de techniques d'optimisation adéquates (compilation au lieu d'interpétation, utilisation non systématique de la programmation logique,...). De plus, à ce moment, vu l'évolution du matériel, les écoles disposeront sans doute de machines nettement plus performantes qu'actuellement.

3.5. La connaissance théorique envisagée.

Remarque : La théorie que le programme connaît peut être lue à la page 31 des annexes et se trouve dans le fichier purethéo.rie.

3.5.1. Présentation de la théorie.

Comme je l'ai signalé auparavant au point **3.2**, j'ai dû réduire la matière et, après réflexion, j'ai décidé d'envisager les problèmes suivants :

Dans une première partie, chaque exercice consistera en la caractérisation de la position l'une par rapport à l'autre de deux droites à savoir si elles sont sécantes, parallèles ou perpendiculaires.

Dans la seconde partie, les exercices consisteront en la caractérisation d'un triangle défini par trois points non-alignés(triangle, triangle isocèle, triangle rectangle, triangle équilatéral).

Pour chacun de ces problèmes, on travaillera dans le plan.

Pour arriver à résoudre ces exercices, j'ai décrit les objets suivants (et quelques propriétés) dans la théorie :

- un point
- la distance d'un point à un autre
- une droite
- l'appartenance d'un point à une droite
- l'équation d'une droite passant par 2 points
- la pente d'une droite
- le parallélisme de deux droites
- la perpendicularité de deux droites
- un triangle
- un triangle isocèle
- un triangle équilatéral
- un triangle rectangle.

Pour que cette théorie soit fonctionnelle dans le programme, il fallait qu'elle soit écrite sous forme de règles PROLOG et donc qu'elle constitue une base de règles. Chaque loi théorique a donc dû être exprimée sous forme d'une ou de plusieurs règles PROLOG.

Je dois également dire au lecteur que je n'ai pas envisagé l'aspect symbolique dans la théorie. Cela signifie que les exercices proposés à l'apprenant seront toujours numériques. La justification de ce point est simple : n'ayant pas pensé tout de suite à de tels exercices, j'ai manqué de temps pour permettre leur utilisation.

3.5.2. La théorie sous forme de règles PROLOG : une contrainte ?

Certains me diront sans doute que, pour l'apprenant, le fait de devoir travailler avec le formalisme de PROLOG entraînera plus de travail.

Il est vrai que pour travailler correctement avec un langage comme PROLOG, il faut une certaine expérience et cela ne s'apprend pas du jour au lendemain.

Le fait de devoir travailler avec PROLOG constitue donc bien sûr une contrainte pour l'enseignant comme pour l'apprenant mais il s'agit d'une contrainte actuelle qu'il y aura moyen de supprimer par la suite : ce que j'expliquerai au point **5.4.**

3.5.3. Présentation des règles théoriques PROLOG.

1°. Le point.

En théorie, dans la géométrie analytique plane, un point du plan est représenté par un couple (x,y) où x est l'abscisse et y l'ordonnée. N'ayant pas envisagé l'aspect symbolique, il faut que x et y soient des nombres réels.

2°. La distance entre deux points.

On peut définir de manière mathématique la distance entre deux points (X1,Y1) et (X2,Y2) et cela donne :

$$\text{distance} = [(X2 - X1)^2 + (Y2 - Y1)^2]^{1/2}$$

3°. La droite.

Une droite est définie par une équation aX + bY + c = 0 à deux variables (X et Y). Ce sont donc les trois coefficients a,b et c qui définissent la droite. On a une droite si le coefficient d'une des variables est non nul. De plus, l'aspect symbolique n'ayant pas été envisagé, il faut que les trois coefficients soient des nombres réels.

4°. L'appartenance d'un point à une droite.

Un point (X1,Y1) appartient à une droite définie par les coefficients (a,b,c) si il vérifie l'équation suivante :

$$a*X1 + b*Y1 + c = 0$$

5°. Une équation d'une droite passant par deux points.

Si on dispose de deux points distincts (X1,Y1) et (X2,Y2) et que l'on veut connaître les coefficients (a,b,c) de la droite (et donc son équation) passant par ces deux points, on peut procéder comme suit :

```
a = Y2 - Y1
b = X1 - X2
c = -Y1 * ( X1 - X2) + X1 * (Y1 - Y2)
```

6°. La pente d'une droite.

La pente d'une droite dont l'équation est aX + bY + c = 0 avec le coefficient a différent de 0 est égale à -b / a (l'opposé de b divisé par a). Si a, le coefficient de x, est égal à 0, la pente est égale à

l'infini. Pour des raisons de facilité, j'ai décidé de fixer la pente de la droite à 9999.

7°. Le parallélisme de deux droites.

Deux droites D1 et D2 sont parallèles si leurs pentes sont égales.

Cette définition est correcte et peut toujours être appliquée mais il existe d'autres moyens de voir que deux droites sont paralléles. Il est nécessaire d'exprimer ces différentes manières de voir si deux droites sont parallèles bien que la première suffise en théorie. Cette nécessité vient du fait qu'un apprenant n'utilisera pas toujours la première façon de faire quand l'une des deux dernières règles sera possible. La théorie doit donc, comme je l'ai déjà dit au point **3.4.1**, être conforme à la façon dont l'élève voit la matière.

Actuellement, il existe deux autres règles permettant de vérifier la parallélisme de deux droites :

1°. Si les coefficients de X et de Y d'une des droites sont égaux aux coefficients de l'autre droite multipliés par un même nombre, on a le parallélisme.

2°. Si les coefficients de X et de Y sont les mêmes pour les deux droites, on a également le parallélisme.

8°. La perpendicularité de deux droites.

Deux droites D1 et D2 sont perpendiculaires si :

1°. Le produit de leurs pentes, différentes de 9999, donne -1.

2°. La pente de la droite D1 est égale à 9999 et la pente de la droite D2 est nulle.

Comme pour le parallélisme, il existe d'autres manières de voir que deux droites sont perpendiculaires

et l'apprenant appliquera ces différentes règles quand il le pourra. Il est donc nécessaire de les exprimer pour les raisons citées au point 3.4.1.

Dans l'état actuel du prototype, il existe deux autres règles permettant de vérifier la perpendicularité de deux droites :

1°. Si les coefficients de X et de Y des deux droites sont permutés et l'un des deux coefficients d'une des droites a le signe contraire du même coefficient de l'autre droite, les deux droites sont perpendiculaires.

2°. Si les coefficients de X et de Y des deux droites permutés sont proportionnels au signe près, les deux droites sont perpendiculaires.

9°. Le triangle.

Un triangle est défini par trois points non-alignés c'est-à-dire ne faisant pas partie de la même droite.

10°. Le triangle isocèle.

Un triangle est isocèle si deux de ses côtés ont la même longueur.

11°. Le triangle équilatéral.

Un triangle est équilatéral si tous ses côtés ont la même longueur.

12°. Le triangle rectangle.

Un triangle est rectangle si deux de ses côtés sont perpendiculaires.

3.6. Les fonctionnalités offertes.

On peut grouper les différentes fonctionnalités possibles du prototype selon leur activité. Voici une liste de fonctionnalités regroupées en classes selon leur appartenance à l'analyseur ou à la description de la stratégie.

Comme je l'ai déjà dit à plusieurs reprises, le but de ce mémoire n'était pas d'arriver à un didacticiel utilisable tel quel mais bien de montrer la faisabilité d'un tel didacticiel. Dès lors, je me suis très peu soucié du caractère ergonomique des fonctionnalités.

3.6.1. Les fonctionnalités liées à l'analyse d'une réponse

3.6.1.1. Trouver la réponse à un exercice.

C'est grâce à cette fonctionnalité que le programme trouve une réponse d'un exercice et mémorise un chemin emprunté pour y arriver. Cette mémorisation sera utilisée par presque toutes les autres fonctionnalités. Cette mémorisation permettra aux autres fonctionnalités d'appliquer la stratégie décrite par le professeur. Cette fonctionnalité est, comme je le montrerai au point **3.7.1**, indépendante de la matière envisagée.

3.6.2. Les fonctionnalités liées à l'application de la description de la stratégie

Dans la version actuelle du prototype, la mémorisation du chemin de résolution se fait dans une liste qui peut être exploitée par les trois fonctionnalités suivantes :

3.6.2.1. Vérifier si la réponse est correcte et terminale.

L'analyseur regarde si la réponse fournie par l'apprenant est équivalente à la réponse finale trouvée par le programme. Si c'est le cas, on considère que la réponse de l'apprenant est correcte et terminale. Si, au

contraire, les deux réponses ne sont pas les mêmes, l'analyseur va consulter la liste des différentes réponses intermédiaires qu'il a trouvées. Si il trouve une réponse intermédiaire qui correspond à celle de l'apprenant, l'analyseur retient la règle que le programme a utilisée pour continuer. Cette mémorisation sera utile au moment où le didacticiel voudra aider l'apprenant dans son processus de résolution.

3.6.2.2. Analyser la réponse de l'apprenant jusqu'à un certain degré de difficulté.

Au pont 2.7.6, j'ai parlé de la difficulté d'un chemin de résolution. Dans le prototype, chaque règle a un degré de difficulté et c'est sur ce degré que la fonctionnalité va travailler.

Par cette fonctionnalité, on offre la possibilité suivante au professeur : il peut vérifier la connaissance et la bonne application des règles dont le degré de difficulté est supérieur au degré qu'il a spécifié.

Pour éviter que l'apprenant ne doive réexpliquer trop de fois une même règle appliquée dans son processus de résolution, j'ai pensé à utiliser un nombre maximum. Quand l'apprenant a montré ce nombre maximum de fois (au cours du même exercice) qu'il connaissait et pouvait appliquer une règle, on ne lui demandera plus de l'expliquer. Ce nombre maximum pourra être modifié par le professeur. On voit ici la nécessité d'un historique : en effet, le nombre maximum ne joue que sur un exercice alors que si on sait appliquer une règle dans un exercice, il n'y a pas de raisons que l'on ne sache pas l'appliquer dans un autre exercice.

3.6.2.3. Vérifier si l'apprenant a utilisé une règle précise.

Remarque : Les règles ont été numérotées dans le prototype.

Le professeur peut, par cette fonctionnalité, voir si l'apprenant a utilisé une régle particulière. Le

professeur décide de la règle en donnant le numéro de celle-ci.

3.6.3. Les fonctionnalités liées à l'interfaçage avec l'apprenant

Il existe ici une multitude de possibilités mais je me suis limité à quelques fonctionnalités qui me semblaient essentielles dans un premier temps.

3.6.3.1. Montrer à l'apprenant que la règle utilisée n'est pas applicable.

Cette fonctionnalité se déroule en deux temps et uniquement si la réponse de l'apprenant apparaît incorrecte aux yeux du programme.

1°. Le programme demande à l'apprenant la règle qu'il a appliquée pour arriver à ce résultat.

2°. Il montre ensuite à l'apprenant la partie de régle prouvant qu'elle n'est pas applicable.

3.6.3.2. Demander à l'apprenant si il désire une explication.

En cas de réponse non terminale, cette procédure demandera à l'apprenant si il désire ou non une explication pour continuer. Si la réponse est oui, le programme appliquera la fonctionnalité décrite au point **3.6.3.3.** Par contre, si elle est négative, le programme laissera l'apprenant se débrouiller.

3.6.3.3. Proposer une explication à l'élève.

Comme je l'ai signalé au point **3.6.2.1**, en cas de réponse intermédiaire, le programme retient la règle qu'il a utilisée pour continuer.

L'explication donnée à l'élève consistera en la présentation de la règle à appliquer pour progresser d'une étape dans le processus de résolution.

3.6.3.4. Féliciter l'élève.

Dans la version actuelle, en cas de bonne réponse, l'interpreteur félicite l'apprenant avant de passer à l'application des fonctionnalités **3.6.2.2** et **3.6.2.3.**

3.6.4. Les fonctionnalités liées à la modification de la description de la stratégie d'apprentissage

3.6.4.1. Initialiser le degré de difficulté critique.

Au point 3.6.2.2., j'ai parlé d'un degré de difficulté critique. Il faudra vérifier la compréhension des règles qui ont un degré supérieur à cette valeur critique. La fonctionnalité permet au professeur d'initialiser ce degré critique. Si il ne veut pas de degré critique, il donne, dans la version actuelle, la valeur 100 comme réponse.

3.6.4.2. Donner le numéro de la règle à vérifier.

Au point 3.6.2.3., j'ai parlé d'une règle qu'il faudra vérifier. Cette fonctionnalité permet au professeur de donner le numéro de cette règle. Si il ne veut pas qu'il y ait une règle à vérifier, il donne, dans la version actuelle, la valeur 0 comme réponse.

Ces deux fonctionnalités montrent la puissance du système : en effet, il suffit, par exemple, que le professeur change le degré de difficulté critique pour que la stratégie soit entièrement modifiée et ce changement se fait en invoquant une fonctionnalité.

3.7. Spécifications des procédures réalisées.

Dans cette partie, je vais expliquer les procédures(1) qui me paraissent les plus importantes pour le didacticiel.

3.7.1. La procédure de mémorisation du chemin de résolution

Cette procédure est, selon moi, la plus importante de toutes. En effet, elle mémorise le chemin de résolution et c'est sur cette mémorisation que les autres procédures vont se baser pour réaliser leurs objectifs.

Remarque : L'underscore qui se trouve avant les variables est nécessaire dans le BIM-Prolog.

Dans le programme, il s'agit de la procédure *trac* dont voici le code :

```
trac(true,[],_L,_D) :- !.
trac((_A,_B),_L,_Lr,_D) :- !,trac(_A,_L,_L1,_D),
                            trac(_B,_L1,_Lr,_D).
trac(not _A,_L,_Lr,_D) :- not trac(_A,_L,_Lr,_D).
trac(!,_L,_L,_D).
trac(_A,_L,_Lr,_D) :- syst(_A),_A,!,
                      ajout(_L,[[_A,'0','0',_D]],_Lr).
trac(_A,_L,_Lr,_D) :- clause(_A,_B),
                      numero((_A :- _B),_Y),
                      degre(_Y,_Z),
                      ajout(_L,[[_A,_Y,_Z,_D]],_L1),
                      _D1 is (_D+1),
                      trac(_B,_L1,_Lr,_D1).
trac(_A,_L,_Lr,_D) :- syst(_A),not _A,fail.
```

Il s'agit d'une procédure que j'ai trouvée dans [SHAPIRO] et que j'ai quelque peu adapté au problème qui me concernait.

---

(1) Je parle ici de procédure au sens PROLOG du terme.

1°. But de la procédure.

La procédure construit une liste de manière incrémentale. Cette liste contient des sous-listes de 4 éléments :

- le premier représente le but présent à prouver,
- le second, le numéro de la règle utilisée pour prouver ce but,
- le troisième, le degré de difficulté de cette règle,
- le dernier représente la profondeur du but à prouver. Par profondeur, j'entends le niveau auquel se situe le prédicat à prouver. Le principe d'incrémentation de cette profondeur est le suivant : si la tête d'une règle a la profondeur X, les prédicats formant la queue de cette règle auront la profondeur X + 1.

2°. Les arguments de la procédure.

La procédure a 4 arguments :

- le premier représente le but courant à prouver,
- le second est la liste de départ,
- le troisième argument est cette liste à laquelle on a ajouté en queue la sous-liste créée par cette procédure pour prouver le but courant,
- le dernier argument est la profondeur actuelle.

3°. Les procédures utilisées.

La procédure fait appel à six autres procédures : clause, fail, syst, ajout, numero, degre. Les deux premières sont des procédures internes au BIM-Prolog : ici, clause(A,B) donne la queue(B) de la règle dont A est la tête et fail rate toujours. J'ai réalisé les quatre autres procédures :

- syst(A) vérifie si son argument est un prédicat-système,
- ajout(A,B,C) crée une liste C en ajoutant à la fin de la liste A la liste B,

- numero(A,B) donne dans l'argument B le numéro de l'argument A qui est une clause,
- degre(A,B) donne dans l'argument B le degré de difficulté associé à la règle ayant pour numéro l'argument A.

Les clauses faisant appel au foncteur syst (5 et 7) permettent de gérer les prédicats-systèmes qui ne sont pas définis dans la base de règles du didacticiel. En adaptant la procédure syst (voir fichier sys page 20 des annexes), on arrivera à une portabilité totale de la procédure *trac*. Cela signifie qu'elle pourra être utilisée quelle que soit la matière à enseigner pour autant que l'on ait respecté ce qui est expliqué au point 5.1.

3.7.2. La procédure de vérification de la réponse.

Cette procédure (*verifrep*) n'est pas parfaite mais je tiens quand même à expliquer la manière dont elle travaille.

En voici le code :

```
verifreponse(_X,_L,_F,_I,_T,_P) :- parcoursreponse(_X,_L,_T),
                                   deuxieme(_L,_T),!,
                                   _F = 'O'.
verifreponse(_X,_L,_F,_I,_T,_P) :- deuxieme(_L,_D),
                                   premier(_U,_D),trt(_U,_M),
                                   _X== _M,!,_F='O'.
verifreponse(_X,_L,_F,_I,_T,_P) :- parcoursreponse(_X,_L,_T),
                                   not (_T == []),!,_F = 'F',
                                   _I = 'O',
                                   cherpremprec(_T,_P,_L).
verifreponse(_X,_L,_F,_I,_T,_P) :- _F = 'F',_I = 'F'.
```

1°. But de la procédure.

Cette procédure vérifie si la réponse donnée par l'apprenant se trouve dans la liste construite par la procédure *trac*. Grâce à la procédure trt, la réponse de l'apprenant ne doit plus être exactement la même que celle trouvée par le système.

2°. Les arguments de la procédure.

La procédure a 6 arguments :

- le premier représente la réponse de l'apprenant.
- le second argument est la liste construite par la procédure *trac*.
- le troisième argument est initialisé à "O" si la réponse de l'apprenant est la réponse finale que le programme a trouvé. Cet argument est mis à "F" si la réponse de l'apprenant ne correspond pas du tout à la réponse du programme.
- le quatrième argument est initialisé à "O" si la réponse de l'apprenant correspond à une des réponses intermédiaires trouvées par le programme. Cet argument prend la valeur "F" si ce n'est pas le cas.
- le cinquième argument est une liste de quatre éléments qui a été initialisée par la procédure parcoursreponse.
- le dernier argument est également une liste à quatre éléments qui sera initialisée par la procédure cherpremprec.

3°. Les procédures utilisées.

La procédure fait appel à cinq autres procédures : parcoursreponse, deuxieme, premier, trt, cherpremprec. On peut trouver l'explication précise de ces procédures dans les annexes mais voici, en bref, ce qu'elles font :

- parcoursreponse(A,B,C) vérifie si l'argument est la tête d'une des sous-listes contenues dans l'argument B. Si c'est le cas, l'argument C prend la valeur de cette sous-liste; sinon on donne à C la valeur d'une liste vide.
- deuxieme(A,B) donne dans l'argument B le second élément de la liste contenue dans l'argument A.
- premier(A,B) donne dans l'argument A le premier élément de la liste contenue dans l'argument B.
- trt(A,B) donne l'argument B qui résulte d'un traitement sur l'argument A.

- cherpremprec(A,B,C) donne dans l'argument B la première sous-liste (de la liste représentée par l'argument C) ayant comme quatrième argument une valeur inférieure de 1 à celle que retient le quatrième argument de la sous-liste représentée par l'argument A.

Chapitre 4 : Mode d'emploi du sous-système.

4.1. Que peut-on faire avec le sous-système ?

Le sous-système implémenté offre au départ trois possibilités à l'utilisateur :

1°. Lire une règle théorique dont il connaît le numéro.
2°. Lire l'entièreté des règles théoriques.
3°. Résoudre un exercice.

La description de ces fonctionnalités se fera dans les points suivants.

4.2. Comment lancer le sous-système ?

La première et la seule chose à faire pour lancer le programme est de taper l'instruction "mémoire" suivie de la touche RETURN. Ensuite, l'utilisateur attend quelques instants avant de voir apparaître l'écran de départ. Avant de voir apparaître cet écran, on passera dans l'environnement BIM-PROLOG et l'utilisateur verra une suite de mots défiler; ce sont les fichiers qui sont compilés. Le premier écran qui s'adresse à lui a la forme suivante :

Utilisation d'un didacticiel

1. Lire une règle.

2. Lire toutes les règles.

3. Faire un exercice.

4. Quitter

Ta réponse :

Il faut répondre en donnant un chiffre de 1 à 4. Pour permettre à l'utilisateur de confirmer son choix en tapant la touche RETURN, j'ai dû faire la chose suivante : la réponse doit être suivie d'un blanc et d'un point. Cette remarque vaut pour tous les endroits où l'utilisateur doit taper une réponse : il doit toujours terminer sa réponse par un point et si le dernier caractère de celle-ci est un chiffre, il est nécessaire de laisser un blanc avant le point. Cette contrainte est due à l'utilisation de PROLOG et n'existera donc plus quand l'interprèteur-analyseur présenté au point **5.4** aura été implémenté.

Quand on quitte le programme, on quitte également l'environnement BIM-PROLOG. Pour relancer le programme, on recommencera donc la même série d'instructions.

4.3. La fonctionnalité "lire une règle".

Quand, suite à la présentation du premier écran, l'utilisateur a choisi l'option 1, il voit apparaître l'écran suivant :

Quelle règle désires-tu lire ?

Ta réponse :

L'utilisateur doit alors donner le numéro de la règle qu'il désire lire. Comme je l'ai signalé au point **4.2.**, la réponse devra être suivie d'un blanc, d'un point et d'un RETURN. Après avoir fait cela, l'utilisateur verra apparaître la règle demandée dans le formalisme PROLOG standard et pour revenir à l'écran de départ, il devra taper n'importe quelle touche.

Cette fonctionnalité est, en principe, destinée au professeur qui peut voir si la règle ayant tel numéro est bien la règle dont il désire savoir si l'apprenant la connaît.

4.4. La fonctionnalité "lire toutes les règles".

Après avoir choisi l'option 2 de l'écran de présentation, l'utilisateur voit apparaître toutes les règles une à une. L'affichage d'une nouvelle règle se fait en tapant n'importe quelle touche.

Cette fonctionnalité permet à l'utilisateur de visualiser toutes les règles en une seule fois.

De nouveau, il me semble que c'est le professeur qui utilisera le plus souvent cette option. Grâce à elle, il pourra trouver le numéro d'une règle particulière.

4.5. La fonctionnalité "Faire un exercice".

Cette fonctionnalité est la plus importante puisque c'est par celle-ci que l'apprenant va pouvoir vérifier sa connaissance de la matière.

Quand l'utilisateur aura choisi cette option, le programme lui posera successivement les questions suivantes :

Quel est le degré de difficulté au dessus duquel il faut poser des questions ?

Ta réponse :

Quelle est la règle à vérifier ?

Ta réponse :

Quel est l'exercice à poser ?

Ta réponse :

Ces trois questions sont destinées au professeur qui peut ainsi voir le degré de connaissance de l'apprenant d'une partie de la matière. En modifiant la réponse d'une des questions, il change la partie de matière. On peut aussi penser à ceci : le professeur établit un programme à

suivre pour chaque apprenant; dans ce programme, figure les différentes réponses successives à donner aux questions.

Pour les raisons déjà expliquées, l'utilisateur aura soin de répondre de la manière suivante : un nombre, un blanc et la touche RETURN.

Quand il aura répondu aux trois questions, le programme proposera l'exercice voulu à l'apprenant. Il lui demandera de donner sa réponse. Dès que le programme a proposé l'exercice à l'élève, il cherche la solution et mémorise le chemin qu'il a emprunté.

Suite à l'introduction de sa réponse par l'apprenant, le programme vérifiera si cette réponse est correcte, finale, intermédiaire... Peuvent alors se présenter différents cas de figures :

1°. Si elle est correcte et finale, le programme félicite l'apprenant pour sa réponse et passe à l'étape suivante.

2°. Si elle est correcte mais intermédiaire, le programme le signalera à l'apprenant et lui demandera si il désire une explication pour continuer. Si il accepte, le programme la lui donnera. Si l'apprenant refuse, le programme le laissera continuer. Dans les deux cas, l'apprenant devra donner une autre réponse que le programme traitera de la même manière.

3°. Si elle est incorrecte, le programme demande à l'apprenant une règle qu'il a utilisée pour arriver à ce résultat et essaye de lui montrer que la règle utilisée n'est pas applicable. Ensuite, on redemande à l'apprenant une réponse qui suit le même processus.

Si la réponse de l'apprenant est correcte et finale, on passe à la vérification des règles ayant un degré de difficulté supérieur au degré critique initialisé plus haut. Dans cette vérification, l'apprenant devra donner les règles appliquées pour prouver différents buts.

Après cela, on demande à l'apprenant si il désire continuer avec les mêmes paramètres. Si c'est le cas, on demande l'exercice à poser et on reprend le processus expliqué. Si ce n'est pas le cas, on revient à l'écran de départ.

Chapitre 5 : Les améliorations et modifications futures.

Les améliorations et les modifications, qui vont être expliquées maintenant ne constituent pas une liste exhaustive et demanderaient, pour leur réalisation, un temps considérable dont je ne disposais pas.

Néanmoins, pour rendre le didacticiel pratiquement utilisable, il me semble que ces modifications et ces améliorations doivent être effectuées.

5.1. Un changement de stratégie au niveau de l'analyse.

Comme je l'ai déjà dit au point **3.3**, la méthode de travail de l'analyseur de réponse que j'ai choisie n'est sans doute pas la meilleure; cette méthode ne permet pas, par exemple, de reconnaître une démarche de l'apprenant comme étant correcte si la réponse donnée n'est pas une bonne réponse.

Il faudra donc changer cette méthode de travail et la transformer en ceci : l'analyseur attend la réponse de l'apprenant et étudie celle-ci sur base de la description de la matière. L'analyseur ne tracera donc plus un chemin de résolution sur lequel il se basera pour étudier la réponse de l'apprenant. Il devra, en quelque sorte, retrouver le chemin suivi par l'apprenant, en connaissant le départ et la fin de ce chemin.

5.2. Une extension de la matière.

Le prototype réalisé ne comporte que peu de règles. Par conséquent, pour qu'il soit une aide réelle au professeur, il faudrait bien évidemment étoffer cette théorie. Cette extension de la théorie n'est, dans l'ensemble, pas très compliquée à réaliser. Il faut, pour cela, effectuer les actions suivantes :

1°. Transformer les règles théoriques en règles PROLOG et les ajouter dans le fichier contenant la théorie. C'est sans doute l'action la plus difficile à réaliser mais en travaillant méthodiquement et en

ayant une connaissance adéquate du PROLOG, il est possible d'y arriver. Il faut également se rendre compte que la matière doit être conforme à la manière dont l'apprenant la perçoit. Cet aspect est très important et cela nécessitera, selon moi, un certain temps avant d'arriver à une vue conforme à celle des élèves.

Il faut, si l'on désire modifier uniquement la matière, effectuer les deux opérations suivantes :

2°. Numéroter ces règles ajoutées.

3°. Donner un degré de difficulté à chacune de ces différentes règles.

Comme on peut le constater, il ne faut pas avoir de très grandes connaissances informatiques pour réaliser cette extension de la matière. Ainsi, le professeur a la possibilité d'étendre cette matière au fur et à mesure de son avancement.

Une fois cette extension de la matière réalisée, le professeur pourra ajouter une série d'exercices se rapportant aux nouvelles règles théoriques.

Il s'agit ici bien entendu de la façon actuelle de procéder mais on peut espérer que dans le futur, la marche à suivre sera simplifiée grâce à un interprèteur qui pourrait transformer la théorie en règles PROLOG et à certaines procédures se chargeant des autres opérations.

5.3. Un comportement tenant compte du passé.

Dans l'état actuel du prototype, dès qu'un exercice est terminé, le programme oublie tout ce qui a été introduit par l'apprenant. Cela ne correspond pas du tout à ce qui se passe lors d'une interaction professeur-apprenant comme je l'ai signalé au point **2.7.2.**

Il faudrait donc pouvoir tenir compte du passé de chaque apprenant et permettre au didacticiel d'exploiter ce passé.

5.4. Un interprêteur-analyseur pour améliorer le dialogue utilisateur / machine.

Le fait de devoir travailler avec le formalisme de PROLOG constitue très certainement une contrainte. C'est une des raisons de la nécessité d'un interprêteur-analyseur. Il existe néanmoins une autre raison à cela : il est actuellement très difficile d'évaluer la correction des réponses de l'apprenant. Actuellement, il faut que les réponses de l'apprenant soient strictement identiques (ou presque) à celles trouvées par le programme alors que souvent, il existe différentes manières de formuler une réponse.

Cet interpréteur permettrait non seulement de faire la transformation des réponses de l'apprenant en leur correspondant PROLOG mais aussi au professeur de décrire sa stratégie d'apprentissage dans un langage plus simple que le PROLOG. Ainsi, cet interpréteur cachera à l'utilisateur l'emploi du PROLOG comme brique de base du programme et permettrait aux utilisateurs de ne pas devoir connaître ce langage.

Le langage utilisé pour exprimer les réponses et la description de la stratégie sera dépendant de la matière. En effet, il ne serait pas pertinent que l'on choisisse un même langage pour toutes les matières. Voici un exemple montrant cette non-pertinence : si on choisit le français comme langage et que l'on a comme matière les équations du second degré, est-ce vraiment un bon choix ? Il faut donc adapter le langage à la matière couverte par le didacticiel.

5.5. Un interface graphique.

La matière se prêtant assez bien à la visualisation, il serait bon, selon moi, d'intégrer un interface graphique au didacticiel.

CONCLUSION.

1. Les problèmes rencontrés.

1.1. L'expression de la théorie.

Au départ, j'ai pris les règles théoriques d'un manuel sur la géométrie analytique et je les ai simplement transformées en règles PROLOG. Mais je me suis très rapidement rendu compte que ces règles ne correspondaient pas à la manière dont l'apprenant perçoit la théorie. Or, comme je l'ai déjà dit à différentes reprises, c'est cette manière de voir la théorie qui est importante car deux personnes, face au même problème, ne raisonnent pas forcément de la même façon. Il a donc fallu que j'essaye d'exprimer celle-ci de manière conforme à la façon dont l'élève la perçoit. J'ai essayé de faire cela mais je crois que c'est à l'utilisation du didacticiel que l'on va se rendre compte qu'il existe encore des manquements à ce niveau et c'est à ce moment qu'on pourra corriger.

1.2. Le langage PROLOG.

Comme je l'ai signalé au point **3.2**, je ne connaissais que très peu le langage PROLOG. Il m'a donc fallu un temps d'apprentissage pour arriver à me débrouiller avec PROLOG. Ce temps fut d'autant plus important que je n'avais utilisé que des langages procéduraux.

1.3. L'analyse des réponses.

Un problème important que j'ai rencontré en développant le sous-système est l'analyse de la réponse de l'apprenant. Ce problème est inhérent à tous les didacticiels où la réponse est construite par l'utilisateur.

En effet, comment faire pour ne pas obliger l'apprenant à donner une réponse strictement identique à celle trouvée par le programme ? Les procédures *trt* et

trtinv essayent de résoudre quelque peu ce problème mais ce n'est pas encore suffisant.

Je crois que ce problème est une source de rejet des didacticiels par les enseignants et c'est pourquoi un interprêteur-analyseur dont j'ai parlé au point **5.4** est nécessaire.

2. Une erreur de conception.

Comme je l'ai dit au point **3.3**, la méthode de travail de l'analyseur de réponses n'est pas la meilleure et je crois avoir commis une erreur en choisissant cette méthode plutôt que celle qui attend la réponse de l'apprenant pour travailler. Cette seconde méthode correspond mieux à ce qui se passe lors d'une interaction professeur/apprenant.

3. Un espoir.

J'ai essayé, par ce travail, de montrer la possibilité de construire des didacticiels ayant une bonne qualité pédagogique et n'exigeant pas un travail de conception trop importante. Pour ce faire, on s'appuiera sur un système auteur qui permettra la description synthétique et non plus analytique des dialogues.

J'espère, malgré l'erreur de conception citée au point 2, que le travail effectué dans le cadre de ce mémoire pourra servir de base pour d'autres mémoires et qu'ainsi, le système auteur présenté au chapitre 2 pourra être réalisé.

ANNEXE

Avant de passer aux textes des procédures, je tiens à faire deux remarques :

1°. Les fichiers qui contiennent les procédures PROLOG sont présentés dans l'ordre alphabétique.
2°. Le terme procédure est employé au sens PROLOG.

Le fichier AFFICHEUNE.PRO

1. La procédure affichereg.

Cette procédure permet d'afficher la règle dont le numéro est l'argument de la procédure. Elle réalise sa fonctionnalité en appelant d'autres procédures.

```
affichereg(_N) :- num(_X,_N),regecris(_N,_X).
```

2. La procédure regecris.

Cette procédure écrit le message prévu et instancié à l'écran.

```
regecris(_N,_X) :- write('La regle ayant le numero '),write(_N),
                   write(' est'),nl,nl,write(_X),nl,nl,suiteregle.
```

3. La procédure afficheune.

Cette procédure demande à l'utilisateur quelle règle il veut lire et ensuite l'affiche en faisant appel à d'autres procédures.

```
afficheune :- write('Quelle regle desires-tu lire ?'),nl,read(_N),
              nl,affichereg(_N).
```

4. La procédure suiteregle.

Cette procédure permet à l'utilisateur de subordonner le passage à 'action suivante au fait qu'il doit taper une touche.

```
suiteregle :- write('Tapez une touche pour continuer '),
              prompt(' '),readc(_X), autreprompt,nl,nl.
```

Le fichier AFFICHETOUT.PRO

1. La procédure affichetout.

Cette procédure affiche toutes les règles contenues dans la théorie une à une. Le passage d'une règle à la suivante se fait au rythme de l'utilisateur qui tape une touche pour continuer.

```
affichetout(_X) :- _X =< 30 ,!,affichereg(_X),_X1 is _X + 1,
                   affichetout(_X1).
affichetout(_X).
```

Le fichier BASE.PRO

1. La procédure exeleve.

Cette procédure qui est composée de faits, est utilisée pour stocker les exercices à proposer à l'apprenant. C'est ici (ainsi que dans la procédure *exsys*) qu'il faut ajouter des exercices si on veut poser d'autres exercices à l'élève.

Ce qui est entre quotes est ce qui est présenté à l'apprenant et le deuxième argument (le nombre) représente le numéro de l'exercice.

```
exeleve(('Caractérise les droites définies par [1,2,3],[2,4,6]'),1).
exeleve(('Caractérise les droites définies par [1,2,3],[3,5,0]'),2).
exeleve(('Caractérise les droites définies par [1,2,3],[1,2,5]'),3).
exeleve(('Caractérise les droites définies par [1,2,3][-2,1,5]'),4).
exeleve(('Caractérise la forme définie par [1,0],[0,0],[0,1]'),5).
```

2. La procédure exsys.

Cette procédure est similaire à la procédure *exeleve* mais ici, ce sont les exercices qu'on donnera à la procédure *trac* qui essayera de les résoudre.

Le premier argument est l'exercice en lui-même et le second représente le numéro de l'exercice.

```
exsys(test([1,2,3],[2,4,6]),1).
exsys(test([1,2,3],[3,5,0]),2).
exsys(test([1,2,3],[1,2,5]),3).
exsys(test([1,2,3],[-2,1,5]),4).
exsys(test([[1,0],[0,0],[0,1]]),5).
```

Le fichier DEGRE.PRO

1. La procédure degre.

La procédure degre, qui est composée de faits, représente le degré de difficulté associé à chaque règle théorique.

On peut expliquer une ligne comme suit : le premier argument représente le numéro de la règle théorique et le second, le degré de difficulté associé à cette règle.

Pour voir de quelle règle il s'agit, il faut aller voir dans le fichier numéro.pro.

```
degre(1,1).
degre(2,2).
degre(3,2).
degre(4,3).
degre(5,3).
degre(6,3).
degre(7,3).
degre(8,4).
degre(9,4).
degre(10,4).
degre(11,4).
degre(12,3).
degre(13,4).
degre(14,3).
degre(15,4).
degre(16,4).
degre(17,5).
degre(18,5).
degre(19,5).
degre(20,6).
degre(21,5).
degre(22,5).
degre(23,5).
degre(24,0).
degre(25,0).
degre(26,0).
degre(27,0).
degre(28,0).
degre(29,0).
degre(30,0).
```

Le fichier ESSAI.PRO.

Il s'agit d'un fichier de commandes qui sera exécuté lorsque l'utilisateur désirera commencer sa session de travail.

Les deux premières lignes consistent à faire en sorte que les procédures PROLOG soient "compilées". La troisième ligne consiste à appeler la procédure "presente" expliquée ailleurs.

```
?- [facile].
?- premier.
?- presente.
```

Le fichier EXER.PRO.

1. La procédure exercice.

Cette procédure est invoquée chaque fois que l'apprenant désire faire un exercice.

Le premier argument représente le numéro de l'exercice à effectuer; le second, le degré de difficulté au dessus duquel il faut vérifier l'application des règles et le troisième argument représente le numéro de la règle à vérifier.

```
exercice(_N,_D,_K) :- exeleve(_X,_N),exsys(_Y,_N),propose(_X),
                      trac(_Y,[],_L,0),saisiereponse(_R),
                      verifreponse(_R,_L,_F,_I,_T,_P),
                      !,verification(_R,_L,_F,_I,_T,_P,_D,_K,_N).
```

2. La procédure verification.

Cette procédure prend les comportements adéquats en fonction de la correction ou non de la réponse de l'apprenant. La première règle décrit ce qui se passe en cas de réponse finale correcte; la seconde règle, ce qui se passe en cas de réponse correcte non terminale et la dernière s'occupe des réponses incorrectes.

Description des arguments :

- Le premier représente la réponse fournie par l'apprenant.
- Le second contient la liste produite par la procédure *trac*.
- Le troisième argument contient un "O" si la réponse de l'élève est la réponse finale trouvée par la procédure *trac* et "N" sinon.
- Le quatrième argument contient un "O" si la réponse de l'élève est une des réponses intermédiaires trouvées par la procédure *trac* et "N" sinon.
- Le cinquième argument est une liste qui a été initialisée par la procédure verifreponse. Si la réponse de l'apprenant est une des réponses trouvées par la procédure *trac*, cet argument contient la sous-liste dans laquelle est incluse la réponse. Sinon, cette liste est vide.
- Le sixième argument est également une liste initialisée par la procédure cherpremprec. Si la réponse de l''élève

fait partie des réponses trouvées par la procédure *trac*, cette liste contient une sous-liste de la liste construite par *trac*; cette sous-liste a comme dernier argument (la profondeur) une valeur inférieure de 1 à la dernière valeur de la sous-liste contenue dans le cinquième argument.
- Les septième, huitième et neuvième arguments représentent respectivement le degré de difficulté critique, le numéro de la règle à vérifier et le numéro de l'exercice à résoudre.

```
verification(_A,_B,_C,_D,_E,_F,_G,_H,_I) :- _C=='O',!,
                                             felicitations,
                                             difficulte(_B,_G),
                                             utilise(_B,_H).
verification(_A,_B,_C,_D,_E,_F,_G,_H,_I) :- _D =='O',!,repint,
                                             eclairci(_F),
                                             saisiereponse(_X),
                                verifreponse(_X,_B,_C1,_D1,_E1,_F1),
                            verification(_X,_B,_C1,_D1,_E1,_F1,_G,_H,_I).
verification(_A,_B,_C,_D,_E,_F,_G,_H,_I) :- trtinv(_A,_K,_I),
                                             mauvaise(_K,[],_L,0),
                                             repfausse,!,
                                             trouvef(_L,_Y),
                                             propfaux(_K),
                                             justifi(_Y),
                                             saisiereponse(_X),
                                verifreponse(_X,_B,_C1,_D1,_E1,_F1),
                            verification(_X,_B,_C1,_D1,_E1,_F1,_G,_H,_I).
```

3. La procédure felicitations.

Cette procédure, comme son nom l'indique, félicite l'élève en cas de bonne réponse.

```
felicitations :- nl, write('Tres bonne reponse'),nl.
```

4. La procédure repfausse.

Cette procédure signale à l'apprenant que sa réponse semble incorrecte au programme.

```
repfausse :- nl,write('Ta reponse me semble incorrecte'),nl.
```

5. La procédure justifi.

Cette procédure écrit à l'écran son argument suivi du bout de phrase "est faux.".

```
justifi(_X) :- nl,write(_X),write(' est faux.'),nl.
```

6. La procédure repint.

Cette procédure signale à l'apprenant que sa réponse n'est pas terminale.

```
repint :- nl,write('Ta reponse n''est pas finale'),nl.
```

7. La procédure propose.

Cette procédure affiche son argument à l'écran.

```
propose(_X) :- write(_X).
```

8. La procédure trtinv.

La procédure permet de retrouver les arguments de l'exercice proposé à l'élève si il a répondu de manière courte c'est-à-dire sans redonner ces arguments.

```
trtinv(parallele,parallele(_X,_Y),_N) :- exsys(test(_X,_Y),_N).
trtinv(perpendiculaire,perpendiculaire(_X,_Y),_N) :-
                                        exsys(test(_X,_Y),_N).
trtinv(coupe,coupe(_X,_Y),_N) :- exsys(test(_X,_Y),_N).
```

9. La procédure propfaux.

Cette procédure montre à l'apprenant que sa réponse est incorrecte. L'argument de la procédure constitue la réponse de l'apprenant.

```
propfaux(_X) :- clause(_X,_R),nl,write('Pour prouver '),write(_X),
                nl,write('on utilise '),write(_R),nl,write('Or,').
```

Le fichier FACILE.PRO.

Ces deux procédures ont un rôle de compilation (*premier*) ou de recompilation (*apres*) des différentes procédures se trouvant dans les fichiers nommés.

```
premier :- [traceur,outil,degre,sys,theorie,base,exer,verrep,
            propexpl,numero,num,saisierep,prof,presente].

apres :- [-traceur,-outil,-degre,-sys,-theorie,-base,-exer,
          -verrep,-propexpl,-numero,-num,-saisierep,-prof,
          -presente].
```

Le fichier MAUVAISE.PRO.

La procédure mauvaise travaille de la même manière que la procédure trac. Elle construit une liste de sous-listes de manière incrémentale.

Son premier argument représente le but courant à prouver; le second argument contient la liste de départ. Le troisième argument représente la nouvelle liste qui est le résultat du collage de la sous-liste construite par la procédure à la fin de la liste de départ. Le dernier argument représente la profondeur du but courant.

La sous-liste construite contient les mêmes éléments que celle construite par la procédure trac (décrite au point 3.7.1.) à une exception près : si on ne peut prouver le but courant, on donne aux deuxième et troisième arguments la valeur 0 et au quatrième argument, la valeur "F". Dès ce moment, la procédure s'arrête.

```
mauvaise(true,[],_L,_D) :- !.
mauvaise((_A,_B),_L,_Lr,_D) :- !,mauvaise(_A,_L,_L1,_D),
                               mauvaise(_B,_L1,_Lr,_D).
mauvaise(not _A,_L,_Lr,_D) :- not mauvaise(_A,_L,_Lr,_D).
mauvaise(!,_L,_L,_D).
mauvaise(_A,_L,_Lr,_D) :- syst(_A),_A,!,
                          ajout(_L,[[_A,'0','0',_D]],_Lr).
mauvaise(_A,_L,_Lr,_D) :- clause(_A,_B),numero((_A :-_B),_Y),
                          degre(_Y,_Z),
                          ajout(_L,[[_A,_Y,_Z,_D]],_L1),
                          _D1 is (_D+1),mauvaise(_B,_L1,_Lr,_D1).
mauvaise(_A,_L,_Lr,_D) :- syst(_A),not _A,!,
                          ajout(_L,[[_A,'0','0','F']],_Lr).
```

Le fichier NUM.PRO.

La procédure *num* est, en quelque sorte redondante avec la procedure *numero* mais elle donne la possibilité à l'utilisateur de visualiser les numéros des règles théoriques dans le formalisme du PROLOG standard et non dans le formalisme du BIM-Prolog c'est-à-dire sans les underscore avant chaque variable. Le premier argument des faits de la procédure représente la règle théorique et le second argument donne le numéro de cette règle théorique.

```
num((point([X,Y]) :- number(X),number(Y)),1).
num((droite([A,B,C]) :- number(A),number(B),number(C),A\==0),2).
num((droite([A,B,C]) :- number(A),number(B),number(C),B\==0),3).
num((pente(9999,[0,B,C])),4).
num((pente(X,[A,B,C]) :- droite([A,B,C]),!,
     X is (-(real(B)/(A)))),5).
num((parallele([X,Y,Z],[X,Y,K]) :- droite([X,Y,Z])),6).
num((parallele(X,Y) :- droite(X),droite(Y),pente(P1,X),
     pente(P2,Y),P1==P2),7).
num((coupe(X,Y) :- droite(X),droite(Y),pente(P1,X),pente(P2,Y),
     P1\==P2),8).
num((perpendiculaire([X,0,Y],[0,W,Z])),9).
num((perpendiculaire([0,W,Z],[X,0,Y])),10).
num((perpendiculaire(X,Y) :- coupe(X,Y),pente(P1,X),pente(P2,Y),
     Z is P1 * P2 , Z == -1.0),11).
num((pointdroite([X,Y],[A,B,C]) :- (A*(X) + B*(Y) + C) =:= 0),12).
num((droitepoint([X1,Y1],[X2,Y2],[A,B,C]) :- A is Y2 - Y1,
     B is X1 - X2,C is (-(Y1) * (X1 - X2)) +(X1 * (Y1 - Y2))),13).
num((dist(D,[X1,X2],[Y1,Y2]) :- I is ((Y1 - X1)**2),
     J is ((Y2 - X2)**2),D is (sqrt(I + J))),14).
num((perpcote(A,B,C) :- droitepoint(A,B,D1),droitepoint(B,C,D2),
     perpendiculaire(D1,D2)),15).
num((triangle([A,B,C]) :- droitepoint(A,C,D),
     not pointdroite(B,D)),16).
num((triangle-isocele([A,B,C]) :- triangle([A,B,C]),dist(D1,A,B),
     dist(D2,B,C),D1=D2,!),17).
num((triangle-isocele([A,B,C]) :- triangle([A,B,C]),dist(D1,A,B),
     dist(D2,A,C),D1=D2,!),18).
num((triangle-isocele([A,B,C]) :- triangle([A,B,C]),dist(D1,B,C),
     dist(D2,A,C),D1=D2,!),19).
num((triangle-equilateral([A,B,C]) :- triangle([A,B,C]),
     dist(D1,A,B),dist(D2,B,C),D1=D2,dist(D3,A,C),D3=D2,!),20).
num((triangle-rectangle([A,B,C]) :- triangle([A,B,C]),
     perpcote(B,A,C),!),21).
num((triangle-rectangle([A,B,C]) :- triangle([A,B,C]),
     perpcote(A,C,B),!),22).
```

```
num((triangle-rectangle([A,B,C]) :- triangle([A,B,C]),
     perpcote(A,B,C),!),23).
num((test(X,Y) :- parallele(X,Y)),24).
num((test(X,Y) :- perpendiculaire(X,Y)),25).
num((test(X,Y) :- coupe(X,Y)),26).
num((test(X) :- triangle-rectangle(X)),27).
num((test(X) :- triangle-equilateral(X)),28).
num((test(X) :- triangle-isocele(X)),29).
num((test(X) :- triangle(X)),30).
```

Le fichier NUMERO.PRO.

La procédure numero (identique ou presque à la procédure num) donne le numéro de chaque règle théorique. Le premier argument de chaque fait est la règle théorique et le second représente le numéro de cette règle.

```
numero((point([_X,_Y]) :- number(_X),number(_Y)),1).
numero((droite([_A,_B,_C]) :- number(_A),number(_B),number(_C),
        _A\==0),2).
numero((droite([_A,_B,_C]) :- number(_A),number(_B),number(_C),
        _B\==0),3).
numero((pente(9999,[_0,_B,_C]) :- !),4).
numero((pente(_X,[_A,_B,_C]) :- droite([_A,_B,_C]),!,
        _X is (-(real(_B)/(_A)))),5).
numero((parallele([_X,_Y,_Z],[_X,_Y,_K]) :- droite([_X,_Y,_Z])),6).
numero((parallele(_X,_Y) :- droite(_X),droite(_Y),pente(_P1,_X),
        pente(_P2,_Y),_P1==_P2),7).
numero((coupe(_X,_Y) :- droite(_X),droite(_Y),pente(_P1,_X),
        pente(_P2,_Y),_P1\==_P2),8).
numero((perpendiculaire([_X,0,_Y],[0,_W,_Z]) :- !),9).
numero((perpendiculaire([0,_W,_Z],[_X,0,_Y]) :- !),10).
numero((perpendiculaire(_X,_Y) :- coupe(_X,_Y),pente(_P1,_X),
        pente(_P2,_Y),_Z is _P1 * _P2 , _Z == -1.0),11).
numero((pointdroite([_X,_Y],[_A,_B,_C]) :-
        (_A*(_X) + _B*(_Y) + _C) =:= 0),12).
numero((droitepoint([_X1,_Y1],[_X2,_Y2],[_A,_B,_C]) :-
        _A is _Y2 - _Y1,_B is _X1 - _X2,
        _C is (-(_Y1) * (_X1 - _X2)) + (_X1 * (_Y1 - _Y2))),13).
numero((dist(_D,[_X1,_X2],[_Y1,_Y2]) :- _I is ((_Y1 - _X1)**2),
        _J is ((_Y2 - _X2)**2),_D is (sqrt(_I + _J))),14).
numero((perpcote(_A,_B,_C) :- droitepoint(_A,_B,_D1),
        droitepoint(_B,_C,_D2),perpendiculaire(_D1,_D2)),15).
numero((triangle([_A,_B,_C]) :- droitepoint(_A,_C,_D),
        not pointdroite(_B,_D)),16).
numero((triangle-isocele([_A,_B,_C]) :- triangle([_A,_B,_C]),
        dist(_D1,_A,_B),dist(_D2,_B,_C),_D1=_D2,!),17).
numero((triangle-isocele([_A,_B,_C]) :- triangle([_A,_B,_C]),
        dist(_D1,_A,_B),dist(_D2,_A,_C),_D1=_D2,!),18).
numero((triangle-isocele([_A,_B,_C]) :- triangle([_A,_B,_C]),
        dist(_D1,_B,_C),dist(_D2,_A,_C),_D1=_D2,!),19).
numero((triangle-equilateral([_A,_B,_C]) :- triangle([_A,_B,_C]),
        dist(_D1,_A,_B),dist(_D2,_B,_C),_D1=_D2,dist(_D3,_A,_C),
        _D3=_D2,!),20).
numero((triangle-rectangle([_A,_B,_C]) :- triangle([_A,_B,_C]),
        perpcote(_B,_A,_C),!),21).
```

```
numero((triangle-rectangle([_A,_B,_C]) :- triangle([_A,_B,_C]),
        perpcote(_A,_C,_B),!),22).
numero((triangle-rectangle([_A,_B,_C]) :- triangle([_A,_B,_C]),
        perpcote(_A,_B,_C),!),23).
numero((test(_X,_Y) :- parallele(_X,_Y)),24).
numero((test(_X,_Y) :- perpendiculaire(_X,_Y)),25).
numero((test(_X,_Y) :- coupe(_X,_Y)),26).
numero((test(_X) :- triangle-rectangle(_X)),27).
numero((test(_X) :- triangle-equilateral(_X)),28).
numero((test(_X) :- triangle-isocele(_X)),29).
numero((test(_X) :- triangle(_X)),30).
```

Le fichier OUTIL.PRO.

1. La procédure appartientliste.

La procédure appartientliste permet de voir si un élément appartient à une liste.

Le premier argument est l'élément en question et le second argument est la liste.

```
appartientliste(_X,[_X|_Y]).
appartientliste(_X,[_Y|_Ys]) :- appartientliste(_X,_Ys).
```

2. La procédure parcoursliste.

La procédure parcoursliste permet de voir si un élément est la tête d'une des sous-listes regroupées dans une liste.

Le premier argument est la liste de sous-listes. Le second argument représente l'élément recherché. Si cet élément est la tête d'une des sous-listes, on donne au troisième argument la valeur "V" et le quatrième argument devient la sous-liste en question. Si ce n'est pas le cas, on donne la valeur "F" au troisième argument.

```
parcoursliste([[_X,_A,_B,_C]],_Y,_O,_S) :- _O = 'F'.
parcoursliste([[_X,_Y,_Z,_C]|_Ys],_Y,_O,_S) :- _O = 'V',
                                                _S = [_X,_Y,_Z,_C].
parcoursliste([[_X,_A,_B,_C]|_Ys],_Y,_O,_S) :-
                                        parcoursliste(_Ys,_Y,_O,_S).
```

3. La procédure parcliste.

La procédure parcliste est très proche de la procédure parcoursliste et permet de voir si un élément est inférieur au troisième élément d'une des sous-listes regroupées dans une liste.

Le premier argument est la liste de sous-listes. Le second argument représente l'élément recherché. Si cet élément est inférieur ou égal au troisième élément d'une des sous-listes, on donne au troisième argument la valeur "V" et le quatrième argument devient la sous-liste en question. Si ce n'est pas le cas, on donne la valeur "F" au troisième argument.

```
parcliste([[_X,_Y,_Z,_C]],_N,_O,_S) :- _Z@<_N,_O='F'.
parcliste([[_X,_Y,_Z,_C]¦_Ys],_N,_O,_S) :- _Z@>=_N,!,_O = 'V',
                                            _S=[_X,_Y,_Z,_C].
parcliste([[_X,_Y,_Z,_C]¦_Ys],_N,_O,_S) :-
                                          parcliste(_Ys,_N,_O,_S).
```

4. La procédure poserquestion.

La procédure poserquestion pose une question à l'utilisateur.

Le premier argument est une liste dont le premier élément sera utilisé dans la question et le second élément contiendra la réponse donnée par l'utilisateur.

```
poserquestion([_Y,_Y1,_Y2,_Y3],_Z) :- nl,
                                       write('Comment as-tu prouve '),
                                       write(_Y),write('?'),nl,
                                       read(_Z).
```

5. La procédure testeregalite.

La procédure testeregalite a pour but de vérifier si la justification de l'apprenant est correcte.

Le premier argument représente la réponse de l'élève, le second, le numéro de la règle qui est la justification correcte. Le troisième argument est une liste dont la tête est l'élément à justifier et le quatrième argument est un compteur; l'apprenant a deux chances pour donner la justification correcte.

```
testeregalite(_X,_Y,_Z,_C) :- _C @=< 2,numero((_X),_A),_A == _Y,!,
                              nl,write('Ta justification est
                              correcte'),nl.
testeregalite(_X,_Y,_Z,_C) :- _C @=< 2 , premier(_K,_Z),
                              findall(_W,clause(_K,_W),_L),
                              appartientliste(_X,_L),!,nl,
                              write('Ta justification est
                              correcte'),nl.
testeregalite(_X,_Y,_Z,_C) :- _C1 is (_C+1),_C1 @=< 2,!,sermon,
                              poserquestion(_Z,_Xs),
                              testeregalite(_Xs,_Y,_Z,_C1).
testeregalite(_X,_Y,_Z,_C) :- _C ==2,num(_K,_Y),presjusti(_K),!.
```

6. La procédure sermon.

La procédure sermon signale à l'apprenant que le programme ne comprends pas sa justification.

```
sermon :- nl,write('Je ne comprends pas ta justification'),nl.
```

7. La procédure presjusti.

La procédure presjusti donne à l'apprenant la justification correcte.

L'argument est la justification.

```
presjusti(_X) :- nl,write('La justification est '),nl, write(_X).
```

8. La procédure ajout.

La procédure ajout ajoute un élément au bout d'une liste.

Le premier argument est la liste de départ, le second est l'élément à ajouter et le troisème est la liste résultante.

```
ajout([],_X,_X).
ajout([_X|_Xs],_Ys,[_X|_Zs]) :- ajout(_Xs,_Ys,_Zs).
```

9. La procédure premier.

La procédure premier permet de trouver le premier élément d'une liste.

Le premier argument est l'élément que l'on cherche, le second est la liste.

```
premier(_X,[_X|_Xs]).
```

10. La procédure deuxieme.

La procédure deuxieme permet de trouver le deuxième élément d'une liste.

Le premier argument est la liste, le second est l'élément que l'on cherche.

```
deuxieme([_X,_Y|_Z],_Y).
```

11. La procédure enleve.

La procédure deuxieme permet d'enlever des sous-listes d'une liste. Le critère d'enlèvement est les trois premiers éléments de ces sous-listes doivent être égaux avec les trois premiers éléments de la sous-liste qu'on veut enlever.

Le premier argument est la liste de départ, le second est la liste qu'on veut enlever et le troisième est la liste résultante.

```
enleve([[_W,_X,_Y,_K]],[_W,_X,_Y,_Z],[]).
enleve([[_W,_X,_Y,_K]],[_A,_B,_C,_D],[[_W,_X,_Y,_K]]).
enleve([[_W,_X,_Y,_K]|_Xs],[_W,_X,_Y,_Z],_Xr) :-
                                    enleve(_Xs,[_W,_X,_Y,_Z],_Xr).
enleve([[_W,_X,_Y,_K]|_Xs],[_A,_B,_C,_D],[[_W,_X,_Y,_K]|_Ys]) :-
                                    enleve(_Xs,[_A,_B,_C,_D],_Ys).
```

12. La procédure diff.

La procédure diff permet de voir si deux éléments sont différents.

```
diff(_X,_Y) :- _X\==_Y.
```

13. La procédure nouvecran.

La procédure nouvecran affiche un certain nombre de lignes blanches à l'écran. C'est l'argument qui détermine ce nombre de lignes blanches.

```
nouvecran(_X) :- _X > 0,!,nl,_X1 = _X-1,nouvecran(_X1).
nouvecran(_X) :- !.
```

Le fichier PRESENTE.PRO.

1. La procédure presente.

La procédure presente affiche un ecran où l'on demande à l'utilisateur de faire un choix d'activité.

```
presente :- nouvecran(20),tab(2),
            write('Utilisation d''un didacticiel'),
            nouvecran(5),tab(2),autreprompt,
            write('1. Initialisation du degre de difficulte '),
            nl,nl,
            tab(2),write('2. Consultation d''une regle '),nl,nl,
            tab(2),write('3. Un exercice '),nl,nl,
            tab(2),write('4. Fin'),nouvecran(5),tab(2),
            read(_R),ensuite(_R).
```

2. La procédure ensuite.

La procédure ensuite fait ce qu'il faut selon la valeur de son argument. Si l'argument vaut 1, elle appele la procédure afficheune et ainsi de suite.

```
ensuite(_X) :- _X==1,!,write('pas fait'),presente.
ensuite(_X) :- _X==2,!,write('pas fait'),presente.
ensuite(_X) :- _X==3,!,nouvecran(31),prof,presente.
ensuite(_X) :- _X==4,!,merci,halt.
ensuite(_X) :- nouvecran(31),write('Un autre choix '),nl,read(_Y),
               ensuite(_Y),!.
```

3. La procédure merci.

La procédure merci est invoquée quand l'utilisateur désire quitter le didacticiel et elle affiche à l'écran un message de remerciement.

```
merci :- nouvecran(15),
         write('Merci d''avoir utilise ce programme'),
         nouvecran(16).
```

Le fichier PROF.PRO.

1. La procédure prof.

Cette procédure appelle d'autres procédures et est utilisée chaque fois qu'on commence un exercice.

```
prof :- posedifficulte(_D),posenumero(_R),suite(_D,_R,'O').
```

2. La procédure suite.

```
suite(_X,_Y,'N') :- !.
suite(_X,_Y,'O') :- poseexer(_N),exercice(_N,_X,_Y),continuer(_R),
                    suite(_X,_Y,_R).
```

3. La procédure posedifficulte.

Cette procédure demande à l'utilisateur le degré de difficulté critique.

```
posedifficulte(_X) :- nl, write('Quel est le degre de difficulte
                      au dessus '), write('duquel il faut poser
                      les questions ?'),nl,read(_X).
```

4. La procédure posenumero.

Cette procédure demande à l'utilisateur le nuéro de la règle à vérifier.

```
posenumero(_X) :- nl,write('Quel est la regle dont il faut
                  verifier '),write('l''utilisation ?')
                  ,nl,read(_X).
```

5. La procédure poseexer.

Cette procédure demande à l'utilisateur le numéro de l'exercice à poser à l'apprenant.

```
poseexer(_X) :- nl,write('Quel est l''exercice a poser ?'),nl,
                read(_X).
```

6. La procédure continuer.

Cette procédure demande à l'utilisateur si l'on continue avec le même degré de difficulté et la même règle.

```
continuer(_X) :- nl,write('Desires-tu continuer avec les memes
                 coefficients'), nl,write(' de difficulte et
                 la meme regle O(ui - N(on ?'),nl,read(_X).
```

7. La procédure autreprompt.

Cette procédure a pour but de définir un nouveau prompt pour le PROLOG. C'est ce prompt qui apparaîtra quand l'utilisateur devra donner une réponse.

```
autreprompt :- prompt('Ta reponse : ').
```

Le fichier PROPEXPL.PRO.

1. La procédure demandeaccord.

La procédure demandeaccord demande à l'apprenant si il désire une explication.

Son argument contiendra la réponse de l'apprenant.

```
demandeaccord(_X):-write('Desires-tu une explication O(ui-N(on ?')
                  ,nl,read(_X).
```

2. La procédure proposeexplication.

La procédure proposeexplication donne une explication utile pour prouver un but en se basant sur son argument.

La première partie de cet argument est le but à prouver et le deuxième élément de l'argument représente le numéro de la règle utilisée pour prouver ce but.

```
proposeexplication([_X,_Y,_Z,_W]) :- num(_A,_Y),
                        write('Pour continuer, je te propose '),
                    write('d''utiliser la regle '),nl,write(_A).
```

3. La procédure eclairci.

La procédure eclairci combine les deux procédures précédentes.

```
eclairci(_T) :-demandeaccord(_X),(_X=='O'),proposeexplication(_T).
eclairci(_T).
```

Le fichier SAISIEREP.PRO.

La procédure saisiereponse demande à l'élève la réponse qu'il peut donner pour l'exercice proposé et lit cette réponse.

```
saisiereponse(_X) :- nl,write('Quelle est ta reponse?'),nl,read(_X).
```

Le fichier SYST.PRO.

Cette procédure a pour but d'identifier les prédicats-systèmes utilisés dans les différentes procédures et ainsi de permettre à la procédure trac de les gérer.

```
syst(number(_X)).
syst(atom(_X)).
syst(_X\==_Y).
syst(diff(_X,_Y)).
syst(_X is _Y).
syst(real(_X)).
syst(_X/_Y).
syst(_X=_Y).
syst(_X==_Y).
syst(_X*_Y).
syst(_X=:=_Y).
syst(_X - _Y).
syst(_X + _Y).
syst(_X**2).
syst(sqrt(_X)).
```

Le fichier THEORIE.PRO.

Ces différentes procédures expriment la connaissance théorique du programme et elles répondent à ce qui a été expliqué au chapitre 3.

La théorie est exprimée dans le formalisme du BIM-Prolog.

```
point([_X,_Y]) :- number(_X),number(_Y).

droite([_A,_B,_C]) :- number(_A),number(_B),number(_C),
                      _A\==0 .
droite([_A,_B,_C]) :- number(_A),number(_B),number(_C),
                      _B\==0 .

pente(9999,[0,_B,_C]):- !.
pente(_X,[_A,_B,_C]) :- droite([_A,_B,_C]),!,
                        _X is (-(real(_B)/(_A))).

parallele([_X,_Y,_Z],[_X,_Y,_K]) :- droite([_X,_Y,_Z]).
parallele(_X,_Y) : droite(_X),droite(_Y),pente(_P1,_X),
                   pente(_P2,_Y),_P1==_P2.

coupe(_X,_Y) :- droite(_X),droite(_Y),pente(_P1,_X),
                pente(_P2,_Y),_P1\==_P2.

perpendiculaire([_X,0,_Y],[0,_W,_Z]):- !.
perpendiculaire([0,_W,_Z],[_X,0,_Y]):- !.
perpendiculaire(_X,_Y) :- coupe(_X,_Y),pente(_P1,_X),
                          pente(_P2,_Y),_Z is _P1 * _P2,
                          _Z == -1.0 .

pointdroite([_X,_Y],[_A,_B,_C]) :-(_A*(_X) + _B*(_Y) + _C) =:= 0 .

droitepoint([_X1,_Y1],[_X2,_Y2],[_A,_B,_C]) :- _A is _Y2 - _Y1,
                                               _B is _X1 - _X2,
                                               _C is (-(_Y1) *
                                               (_X1 - _X2)) +
                                               (_X1 * (_Y1 -_Y2)).

dist(_D,[_X1,_X2],[_Y1,_Y2]) :- _I is ((_Y1 - _X1)**2),
                                _J is ((_Y2 - _X2)**2),
                                _D is (sqrt(_I + _J)).

perpcote(_A,_B,_C) :-droitepoint(_A,_B,_D1),
                     droitepoint(_B,_C,_D2),
                     perpendiculaire(_D1,_D2).
```

```
triangle([_A,_B,_C]) :- droitepoint(_A,_C,_D),
                        not pointdroite(_B,_D).

triangle-isocele([_A,_B,_C]) :- triangle([_A,_B,_C]),
                                dist(_D1,_A,_B),
                                dist(_D2,_B,_C),_D1=_D2,!.
triangle-isocele([_A,_B,_C]) :- triangle([_A,_B,_C]),
                                dist(_D1,_A,_B),
                                dist(_D2,_A,_C),_D1=_D2,!.
triangle-isocele([_A,_B,_C]) :- triangle([_A,_B,_C]),
                                dist(_D1,_B,_C),
                                dist(_D2,_A,_C),_D1=_D2,!.

triangle-equilateral([_A,_B,_C]) :- triangle([_A,_B,_C]),
                                    dist(_D1,_A,_B),
                                    dist(_D2,_B,_C),_D1=_D2,
                                    dist(_D3,_A,_C),
                                    _D3=_D2,!.

triangle-rectangle([_A,_B,_C]) :- triangle([_A,_B,_C]),
                                  perpcote(_B,_A,_C),!.
triangle-rectangle([_A,_B,_C]) :- triangle([_A,_B,_C]),
                                  perpcote(_A,_C,_B),!.
triangle-rectangle([_A,_B,_C]) :- triangle([_A,_B,_C]),
                                  perpcote(_A,_B,_C),!.

test(_X,_Y) :- perpendiculaire(_X,_Y).
test(_X,_Y) :- coupe(_X,_Y).
test(_X,_Y) :- parallele(_X,_Y).

test(_X) :- triangle-rectangle(_X).
test(_X) :- triangle-equilateral(_X).
test(_X) :- triangle-isocele(_X).
test(_X) :- triangle(_X).
```

Le fichier TRACEUR.PRO.

1. La procédure utilise.

Le premier argument est la liste construite par la procédure trac et le second argument est le numéro de la règle.

```
utilise(_L,_N) :- parcoursliste(_L,_N,_X,_Y),not _X == 'F',!,
                  poserquestion(_Y,_Z),testeregalite(_Z,_N,_Y,1),
                  enleve(_L,_Y,_L1),utilise(_L1,_N).
utilise(_L,_N).
```

2. La procédure difficulte.

Le premier argument est la liste construite par la procédure trac et le second argument est le degré de difficulté critique.

```
difficulte(_L,_N) :- parcliste(_L,_N,_X,_Y),not _X == 'F',!,
                     deuxieme(_Y,_K),poserquestion(_Y,_Z),
                     testeregalite(_Z,_K,_Y,1),enleve(_L,_Y,_L1),
                     difficulte(_L1,_N).
difficulte(_L,_N).
```

3. La procédure trac.

Cette procédure permet de réaliser la fonctionnalité 3.6.1.1. Elle a déja été expliquée en détail au point 3.7.1.

```
trac(true,[],_L,_D) :- !.
trac((_A,_B),_L,_Lr,_D) :- !,trac(_A,_L,_L1,_D),
                           trac(_B,_L1,_Lr,_D).
trac(not _A,_L,_Lr,_D) :- not trac(_A,_L,_Lr,_D).
trac(!,_L,_L,_D).
trac(_A,_L,_Lr,_D) :- syst(_A),_A,!,
                      ajout(_L,[[_A,'0','0',_D]],_Lr).
trac(_A,_L,_Lr,_D) :- clause(_A,_B), numero((_A : _B),_Y),
                      degre(_Y,_Z),ajout(_L,[[_A,_Y,_Z,_D]],_L1),
                      _D1 is (_D+1),trac(_B,_L1,_Lr,_D1).
trac(_A,_L,_Lr,_D) :- syst(_A),not _A,fail.
```

Le fichier VERREP.PRO.

1. La procédure parcoursreponse.

La procédure vérifie si un élément est la tête d'une des listes comprises dans une autre liste.

Le premier argument est l'élément que l'on essaye de trouver; le second est la liste et le troisième est un argument qui prendra la valeur de la sous-liste dont la tête est égal à l'élément recherché.

```
parcoursreponse(_X,[[_A,_B,_C,_D]],_T) :- (_T = []).
parcoursreponse(_X,[[_X,_Y,_Z,_W]¦_Xs],_T) :-
                                    (_T = [_X,_Y,_Z,_W]),!.
parcoursreponse(_X,[[_A,_Y,_Z,_W]¦_Xs],_T) :-
                                  parcoursreponse(_X,_Xs,_T).
```

2. La procédure verifreponse.

Cette procédure vérifie si un élément se trouve à une certaine place dans une liste et a déjà été expliquée au point 3.7.2.

```
verifreponse(_X,_L,_F,_I,_T,_P) :- parcoursreponse(_X,_L,_T),
                                   deuxieme(_L,_T),!,_F = 'O'.
verifreponse(_X,_L,_F,_I,_T,_P) :- deuxieme(_L,_D),premier(_U,_D),
                                   trt(_U,_M),_X== _M,!,_F='O'.
verifreponse(_X,_L,_F,_I,_T,_P) :- parcoursreponse(_X,_L,_T),
                                   not (_T == []),!,_F = 'F',
                                  _I = 'O',cherpremprec(_T,_P,_L).
verifreponse(_X,_L,_F,_I,_T,_P) :- _F = 'F',_I = 'F'.
```

3. La procédure cherpremprec.

Cette procédure a pour but de parcourir une liste de sous-listes de 4 éléments et de renvoyer une sous-liste répondant à la condition suivante : il faut que le dernier élément de cette sous-liste ait une valeur inférieure de 1 à la valeur du quatrième élément de la liste contenue dans le premier argument de la procédure.

Le premier argument est une liste qui va permettre d'initialiser la condition; le deuxième sera le résultat de cette procédure et le troisième est la liste qu'on parcourera.

```
cherpremprec([_A,_B,_C,_D],_Y,_Z) :- _X is (_D-1),
                                     parliste(_Z,_X,_Y).
```

4. La procédure trt.

La procédure trt a pour but de traiter certains mots et d'ainsi faciliter le traitement des réponses fournies par l'apprenant. En effet, grâce à cette procédure, si le système a trouvé comme réponse finale "parallèle([1,1,2],[2,2,8])" et que l'apprenant répond "parallèle", on considèrera la réponse comme correcte.

```
trt(parallele(_X,_Y),parallele).
trt(perpendiculaire(_X,_Y),perpendiculaire).
trt(coupe(_X,_Y),coupe).
trt(triangle(_X),triangle).
trt(triangle-equilateral(_X),triangle-equilateral).
trt(triangle-isocele(_X),triangle-isocele).
trt(triangle-rectangle(_X),triangle-rectangle).
```

5. La procédure parliste.

Cette procédure va également parcourir une liste de sous-listes et renvoyer une sous-liste qui pourra avoir les valeurs suivantes :

- si dans une des sous-listes de la liste parcourue, il y en a une dont le dernier élément est égal au second argument de la procédure, on donnera au troisième argument la valeur de cette sous-liste,
- si ce n'est pas le cas, on donne au troisième argument la valeur d'une liste vide.

Le premier argument est la liste qu'on va parcourir. Le second argument est un nombre qui va initialiser les conditions et le troisième est le résultat qui sera soit une liste de 4 éléments si le deuxième argument est égal à la queue de cette liste soit la liste vide si ce n'est pas le cas.

```
parliste([[_A,_B,_C,_D]],_Y,_Z) :- (_Z = []).
parliste([[_A,_B,_C,_Y]¦_X],_Y,_Z) :- _Z = [_A,_B,_C,_Y],!.
parliste([[_A,_B,_C,_D]¦_X],_Y,_Z) :- parliste(_X,_Y,_Z).
```

Le fichier PURETHEO.RIE.

Dans ce fichier, on trouve l'entièreté des règles théoriques dans le formalisme du PROLOG.

```
point([X,Y]) :- number(X),number(Y).

droite([A,B,C]) :- number(A),number(B),number(C),A\==0 .
droite([A,B,C]) :- number(A),number(B),number(C),B\==0 .

pente(9999,[0,B,C]).
pente(X,[A,B,C]) :- droite([A,B,C]),!,X is (-(real(B)/(A))).

parallele([X,Y,Z],[X,Y,K]) :- droite([X,Y,Z]).
parallele(X,Y) :- droite(X),droite(Y),pente(P1,X),pente(P2,Y),
                  P1==P2.

coupe(X,Y) :- droite(X),droite(Y),pente(P1,X),pente(P2,Y),P1\==P2.

perpendiculaire([X,0,Y],[0,W,Z]).
perpendiculaire([0,W,Z],[X,0,Y]).
perpendiculaire(X,Y) :- coupe(X,Y),pente(P1,X),pente(P2,Y),
                        Z is P1 * P2, Z == -1.0 .

pointdroite([X,Y],[A,B,C]) :- (A*(X) + B*(Y) + C) =:= 0 .

droitepoint([X1,Y1],[X2,Y2],[A,B,C]) :- A is Y2 - Y1,
                                        B is X1 - X2,
                                        C is (-(Y1) * (X1 - X2)) +
                                             (X1 * (Y1 - Y2)).

dist(D,[X1,X2],[Y1,Y2]) :- I is ((Y1 - X1)**2),
                           J is ((Y2 - X2)**2),
                           D is (sqrt(I + J)).

perpcote(A,B,C) :- droitepoint(A,B,D1),droitepoint(B,C,D2),
                   perpendiculaire(D1,D2).

triangle([A,B,C]) :- droitepoint(A,C,D),not pointdroite(B,D).

triangle-isocele([A,B,C]) :- triangle([A,B,C]),dist(D1,A,B),
                             dist(D2,B,C),D1=D2,!.
triangle-isocele([A,B,C]) :- triangle([A,B,C]),dist(D1,A,B),
                             dist(D2,A,C),D1=D2,!.
triangle-isocele([A,B,C]) :- triangle([A,B,C]),dist(D1,B,C),
                             dist(D2,A,C),D1=D2,!.
```

```
triangle-equilateral([A,B,C]) :- triangle([A,B,C]),dist(D1,A,B),
                                 dist(D2,B,C),D1=D2,dist(D3,A,C),
                                 D3=D2,!.

triangle-rectangle([A,B,C]) :-triangle([A,B,C]),perpcote(B,A,C),!.
triangle-rectangle([A,B,C]) :-triangle([A,B,C]),perpcote(A,C,B),!.
triangle-rectangle([A,B,C]) :-triangle([A,B,C]),perpcote(A,B,C),!.

test(X,Y) :- perpendiculaire(X,Y).
test(X,Y) :- coupe(X,Y).
test(X,Y) :- parallele(X,Y).

test(X) :- triangle-rectangle(X).
test(X) :- triangle-equilateral(X).
test(X) :- triangle-isocele(X).
test(X) :- triangle(X).
```

BIBLIOGRAPHIE.

[BARR & FEIGENBAUM] : BARR A., FAIGENBAUM E.A., 1982, The Handbook of Artificial Intelligence, William Kaufmann, Inc. Heuristech Press.

[BLOOM] : Le défi des deux sigmas : trouver des méthodes d'enseignement collectif aussi efficaces qu'un précepteur....

[BROWN] : BROWN J.S., 1977. Uses of artificial intelligence and advanced computer technology in education. In R.J.Seidel and M.Rubin (Eds.), Computers and communicatins : Implications for education. New-York : Academic Press, 253-270.

[CARBONELL] : CARBONELL J.R., Mixed-initiative man-computer instructional dialogues. BBN Rep. N°.1971, Bolt Berank and Newman, Inc., Cambridge, Mass.

[DEPOVER] : Contribution à un cadre conceptuel pour un enseignement adaptatif médiatisé par ordinateur : Thèse de doctorat en Sciences Psycho-Pédagogiques présentée par Christian DEPOVER (Université de l'Etat Mons 1985).

[KOFFMAN & BLOUNT] : KOFFMAN E.B., BLOUNT S.E., 1975. Artificial Intelligence and automatic programming in CAI. Artificial Intelligence 6:215-234.

[PAGANO] : R.L.PAGANO, J.M.BARRETO, 1990, Hypertext and the teaching process, January.

[SHAPIRO] : L. STERLING, E. SHAPIRO, 1986, The Art of Prolog : Advanced Programming Techniques, MIT Press.

[TAYLOR] : TAYLOR R.P., 1980, The Computer in the School : Tutor, Tool, Tutee. Teachers' College, Columbia University, Teachers' College Press, New York.